5-2012

New Algorithms for High-Throughput Decoding with Low-Density Parity-Check Codes using Fixed-Point SIMD Processors

Jawone Kennedy
Clemson University, jkenned@clemson.edu

New Algorithms for High-Throughput Decoding with Low-Density Parity-Check Codes using Fixed-Point SIMD Processors

---

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Electrical Engineering

---

by
JaWone Anthony Kennedy
May 2012

Accepted by:
Dr. Daniel L. Noneaker, Committee Chair
Dr. Michael B. Pursley
Dr. Harlan B. Russell
Dr. Shuhong Gao

Abstract

Most digital signal processors contain one or more functional units with a single-instruction, multiple-data architecture that supports saturating fixed-point arithmetic with two or more options for the arithmetic precision. The processors designed for the highest performance contain many such functional units connected through an on-chip network. The selection of the arithmetic precision provides a trade-off between the task-level throughput and the quality of the output of many signal-processing algorithms, and utilization of the interconnection network during execution of the algorithm introduces a latency that can also limit the algorithm's throughput.

In this dissertation, we consider the turbo-decoding message-passing algorithm for iterative decoding of low-density parity-check codes and investigate its performance in parallel execution on a processor of interconnected functional units employing fast, low-precision fixed-point arithmetic. It is shown that the frequent occurrence of saturation when 8-bit signed arithmetic is used severely degrades the performance of the algorithm compared with decoding using higher-precision arithmetic. A technique of limiting the magnitude of certain intermediate variables of the algorithm, the extrinsic values, is proposed and shown to eliminate most occurrences of saturation, resulting in performance with 8-bit decoding nearly equal to that achieved with higher-precision decoding.

We show that the interconnection latency can have a significant detrimental

effect of the throughput of the turbo-decoding message-passing algorithm, which is illustrated for a type of high-performance digital signal processor known as a stream processor. Two alternatives to the standard schedule of message-passing and parity-check operations are proposed for the algorithm. Both alternatives markedly reduce the interconnection latency, and both result in substantially greater throughput than the standard schedule with no increase in the probability of error.

Acknowledgments

I would like to thank Professor Michael B. Pursley, Professor Harlan B. Russell, and Professor Shuhong Gao for serving as members of my committee. I would like to extend special thanks to Professor Daniel L. Noneaker, for his valuable guidance throughout my graduate studies at Clemson University, and his considerable patience and assistance with this dissertation. I would also like to thank the Southern Regional Education Board and Space and the Naval Warfare Systems Center, Atlantic, for their monetary support and guidance of my education and research.

I would especially like to thank my wife, Vanessa, for her encouragement and faith throughout my studies, my family and friends, and most of all Jesus Christ, for without him I know this would have not been possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Message-passing decoding algorithms for low-density parity-check (LDPC) codes [1] are generally well suited to exploit the data-level parallelism (DLP) [2] available in most modern processor hardware architectures. Some embedded general-purpose processors and most mainstream embedded digital signal processors (DSPs) incorporate a single-instruction, multiple-data (SIMD) architecture [2] which provides a modest degree of DLP [3, 4]. The SIMD architecture can be utilized in these instances to achieve the required LDPC decoder throughput for applications with low-to-moderate data rates [5]. In contrast, high-data-rate wireless communication systems such as Wi-Fi (IEEE Std. 802.11n) [6], WiMAX (IEEE Std. 802.16) [7], and Digital Video Broadcast – Satellite – Second Generation (ETSI DVB-S2) [8] require an LDPC decoder implementation that utilizes a high level of DLP [9] while satisfying the stringent limits on power consumption for the system's handheld devices.

The dual performance objectives are commonly achieved with either an application specific integrated circuit (ASIC) [10] or a field-programmable gate array

(FPGA) [11], but an alternative of increasing interest is the use of a SIMD-oriented DSP that is optimized for a high degree of DLP in a low-power, embedded system. Such a processor can provide the high-level-language programmability of a general-purpose DSP but with a much greater computational throughput relative to power dissipation for algorithms that admit high levels of DLP. Two common traits characterize many of them: the availability of multiple data-precision modes for SIMD operations, and a hierarchy of SIMD functional units interconnected by a chip-level network [12].

Most embedded DSPs with SIMD capability support fixed-point saturating arithmetic [2] with two or more data-precision modes operating on packed operands so that the lower-precision mode provides greater computational throughput than the higher-precision mode [13]. (For example, the peak computational throughput with 8-bit SIMD arithmetic can approach twice the peak throughput with 16-bit SIMD arithmetic.) The lower-precision mode can result in poorer outcomes for some signal-processing algorithms (such as some LDPC decoding algorithms [14]), however. The selection of the data-precision mode for SIMD operations thus represents a tradeoff between the algorithm's throughput and the quality of its output, and there is a strong motivation for developing algorithms that produce high-quality results even if implemented with low precision.

Specialized DSPs optimized for high levels of DLP in embedded applications provide SIMD capability in a large number of functional units that are interconnected with an on-chip bus, crossbar, or switched network [15–18]. If several (or all) of the functional units are used concurrently to increase the parallelization of a single instance of a signal-processing algorithm (i.e., a single task), the latency of data transfers through the network can be a significant factor in the execution time of the algorithm [19]. The data-transfer intensity of an algorithm thus constrains the

application-level data rate it can support, and there is a strong motivation for developing algorithms that minimize the need for communications among the functional units.

1.2 Contributions of the research

In this dissertation, we address the impact of both the data precision and the data-transfer latency on the performance of the decoding algorithm for a LDPC code as well as methods to mitigate their impact. Specifically, we focus on the *turbo-decoding message-passing* (TDMP) *algorithm* [20] and the characteristics of the algorithm that affect its performance when implemented on a processor with low-precision, fixed-point saturating arithmetic and on a processor that employs multiple SIMD functional units. The TDMP algorithm is a (merged schedule) layered-decoding belief-propagation algorithm [21] that is readily adapted to differing degrees of partially parallel computation based on the degree of DLP available in the processor. We consider its use with an early termination rule, which increases the throughput of the iterative decoder by allowing it to exit the decoding algorithm prior to the maximum number of allowed iterations if the decoder has produced a valid code word.

The TDMP algorithm achieves performance comparable to the sum-product algorithm (SPA) [1] with about one-half the average number of iterations [22] (and thus results in higher throughput for a given degree of DLP). It also requires less memory than the SPA [20], and it permits simpler data management than the SPA in SIMD-oriented processor architectures organized for a high degree of DLP [14, 19]. Partially parallel implementation of the TDMP algorithm is especially well matched with *architecture-aware LDPC* (AA-LDPC) *codes* [20], which include the LDPC codes specified in each of the wireless communication standards noted above.

We investigate the effect of the maximum number of iterations, the code's rate, and the code's block length on the performance of fixed-point TDMP decoding of LDPC codes using 8-bit saturating, signed arithmetic — the lowest-precision mode available for packed-data SIMD operations on many DSPs. The sensitivity of the TDMP algorithm and the SPA to the data precision is also compared. The offset-min-sum (OMS) approximation [23] is used with each algorithm; it is well suited to efficient implementation on a fixed-point processor, and it achieves good performance without requiring an estimate of the signal-to-noise ratio in the received signal.

It is shown that the TDMP algorithm is more sensitive than the SPA to the precision of fixed-point saturating arithmetic. Indeed, its performance can be degraded significantly if the precision is decreased from 16 bits to 8 bits unless the number of iterations is small, whereas the performance of the SPA is not affected noticeably. The detrimental effect of arithmetic saturation on the performance of the TDMP algorithm increases as the number of iterations is increased, and it is more pronounced with codes of lower rate or greater block length. We investigate a technique to limit the occurrence of saturation in TDMP decoding by constraining the magnitude of extrinsic messages in the algorithm; it results a probability of error with 8-bit arithmetic that is nearly the same as that achieved with optimized 16-bit arithmetic (while requiring a much-reduced decoding time in packed-operand SIMD processors). The additional instructions required to implement the technique result in a negligible increase in the decoding time of a DSP implementation.

We also consider the effect of the data precision on the performance of the TDMP algorithm using the max-quartet (MQ) approximation [20]. The MQ approximation results in better performance than the OMS approximation at a low-to-moderate signal-to-noise ratio, but it requires that the receiver estimate the signal-to-noise ratio. It is shown that while the technique we develop for OMS TDMP decoding

is not sufficient to mitigate the effect of saturation with MQ TDMP decoding, a modification of it is effective and results in a probability of error with 8-bit decoding close to the probability of error with 16-bit decoding.

Finally, we investigate alternatives to the standard schedule of posterior updates used in partially parallel, early-termination TDMP decoding of AA-LDPC codes on a *stream processor* [24]. A stream processor is a highly parallel SIMD architecture with a software-managed, cacheless memory hierarchy that adheres to the stream-processing computing paradigm [24]; it is designed for applications that exhibit a high compute intensity, a high degree of DLP, and producer-consumer locality [25]. A stream processor shares some characteristics with a graphical processing unit (GPU). It provides a more efficient trade-off between SIMD parallelism and power consumption than a GPU, however, by supporting only fixed-point arithmetic for single-task, single-threaded execution, emphasizing greater local data-memory density around the functional units, and by omitting some specialized functions found in the GPU. The stream processor we use an an example for numerical results is the commercial Storm-1 processor system-on-a-chip [15].

We consider two algorithms in which the posterior updates and parity checks are integrated for each subset of the check nodes processed in parallel, in contrast with the standard TDMP schedule in which all the parity checks for an iteration are performed after all the updates. The alternative schedules reduce by one-half the data communications required between the stream processor's functional-unit clusters for each iteration of the decoder compare with the standard schedule. Termination rules which guarantee a valid code word are specified for each integrated update-and-parity-check decoding algorithm. It is shown that properly designed integration of the update and parity-check steps results in significantly higher decoder throughput than the standard schedule of the TDMP algorithm without an increase in the probability

of error.

1.3 Related prior work

The effect of the precision of fixed-point arithmetic on the performance of the SPA has been examined in detail in numerous previous investigations (e.g., [26–30]). The motivating application is typically an ASIC implementation; lower-precision arithmetic permits a smaller, more power-efficient ASIC design. Depending on the LDPC code and the maximum number of iterations used in the decoder, the variants of the SPA achieve performance approaching that of high-precision floating-point arithmetic if they employ fixed-point arithmetic with an appropriately chosen resolution (quantization step size) and between four and six bits of precision in the quantized channel outputs, the extrinsic messages exchanged in the algorithm, and the updated (pseudo-)posterior value determined for each code symbol. Too few bits of precision result in an unacceptably high error floor in the decoder's performance.

Prior investigation of fixed-point decoding is much more limited for the TDMP algorithm than for the SPA. Four-bit precision is employed in an ASIC design [32] which results in no apparent error floor down to a probability of code-word error of 10^{-4} for a rate-1/2 AA-LDPC code with ten iterations of the TDMP algorithm. Fixed-point TDMP decoding using seven-bit precision (described as six bits of magnitude precision in the paper) results in only a small degradation in the probability of bit error compared with floating-point decoding for an example of a rate-3/4 AA-LDPC code [31], though the number of iterations is not specified. No discernible error floor down to a probability of bit error of 10^{-6} is observed with five iterations of the min-sum variant of the TDMP algorithm in an implementation on a graphical processor unit using 8-bit precision for posterior values and 6-bit precision for extrinsic

values for each of two rate-1/2 WiMAX LDPC codes [33].

A larger number of iterations per received word (30) is considered in [34] in which an ASIC design using 8-bit precision for posterior values and 6-bit precision for extrinsic values results in an error floor that begins at a probability of bit error of 10^{-5} for a modified TDMP algorithm and two rate-1/2 WiMAX LDPC codes. A similar error floor occurs with an ASIC decoder employing 15 iterations of the normalized min-sum TDMP algorithm [23] using an unspecified precision for one example of a rate-1/2 WiMAX LDPC code [35]. A fixed-point decoder architecture appropriate for ASIC implementation is considered in [36], and the precision and resolution used for each type of data item in the design is optimized. The decoder exhibits no error floor to a probability of code-word error of 10^{-5} for an example of a rate-3/4 WiMAX code.

One of the two alternative TDMP schedules we consider (the "integrated parity check with stability check") has been pointed out previously [21, 37], though not in the context of implementation on a stream-processor architecture. (It is referred to as an "on-the-fly convergence check" in the latter paper.) No evaluation of its impact on the probability of error or the decoding time is provided in either paper.

Previous research has addressed LDPC decoding with the SPA using floating-point arithmetic on a GPU [38–40]. Low-power, fixed-point stream processors present different decoder design tradeoffs than a floating-point implementation on a GPU, however, and the TDMP algorithm entails different design considerations than the SPA. Fixed-point TDMP decoding on a GPU is considered in [33]. None consider alternatives to mitigate the latency of data transfers between the processor's functional units.

In the numerical results reported in Chapter 8 of this dissertation, the full resources of the stream processor are devoted to decoding one received word at a

time in order to minimize the total buffering and decoding delay. In contrast, the research on LDPC decoding on a GPU is focused on using task-level parallelism in which multiple received words are decoded concurrently in different functional units of the processor. Task-level parallelism can reduce the impact of data-transfer latency on the decoding of each received word and maximize throughput, but it increases the average buffering delay incurred prior to decoding in proportion to the number of concurrently decoded received words.

Chapter 2

Communication System and Measures of Performance

2.1 Communication system

The block-coded digital communication system considered in the dissertation is illustrated in Fig. 2.1. A message source produces a K-bit information word, $\underline{\mathbf{u}} = (u_0, \ldots, u_{K-1})$, that is encoded as an N-bit code word, $\underline{\mathbf{v}} = (v_0, \ldots, v_{N-1})$, by a rate (N, K) linear binary block encoder. The encoder realizes one of the LDPC codes discussed in Chapter 3.

Each code symbol is transmitted on the channel using binary antipodal modulation so that the $i^{\underline{th}}$ channel symbol at the demodulator output is given by

$$s_i = \sqrt{\mathcal{E}_c}\ (-1)^{v_i},\ 0 \leq i \leq N-1,$$

where $\mathcal{E}_c$ is the energy per channel symbol. The channel is an additive white Gaussian noise (AWGN) process so that the real-valued $i^{\underline{th}}$ received symbol at the output of

the demodulator is given by

$$r_i = (s_i + n_i),\ 0 \leq i \leq N-1,$$

where $\{n_0, \ldots, n_{N-1}\}$ are independent and identically distributed (i. i. d.) Gaussian random variables with $E[n_j] = 0$ and $\mathrm{Var}(n_j) = \sigma^2$, for $0 \leq j \leq N-1$. The *signal-to-noise-ratio* (SNR) in the received signal is given by $\mathcal{E}_b/(2\sigma^2)$, where $\mathcal{E}_b = (N/K)\ \mathcal{E}_c$ is the energy per bit of information.

The receiver using each decoding algorithm considered in the dissertation applies a constant amplitude gain G_{AGC} to each received symbol r_i to produce the symbol $\hat{r}_i$. The gain G_{AGC} represents the scaling of the received symbols prior to decoding as a result of automatic gain-control amplification of the received signal (which precedes the demodulator, in practice) or amplitude normalization based on the output of a signal-amplitude estimator (which follows the demodulator), or both. In the dissertation, we assume that ideal amplitude normalization is achieved so that

$$G_{\mathrm{AGC}} = \left(\sqrt{\mathcal{E}_c}\right)^{-1}. \tag{2.1}$$

Consequently,

$$\hat{r}_i = (-1)^{v_i} + \hat{n}_i,\ 0 \leq i \leq N-1,$$

where $\{\hat{n}_0, \ldots, \hat{n}_{N-1}\}$ are i. i. d. Gaussian random variables with $E[\hat{n}_i] = 0$ and

$$\mathrm{Var}(\hat{n}_i) = \hat{\sigma}^2 = \frac{\sigma^2}{\mathcal{E}_c} = \frac{1}{2}\left(\frac{N}{K}\right)\left(\frac{\mathcal{E}_b}{2\sigma^2}\right)^{-1},\ 0 \leq i \leq N-1. \tag{2.2}$$

A subset of the decoding algorithms considered in the dissertation apply no

further scaling to the received symbols so that each scaled received symbol is given by

$$\delta_i = \hat{r}_i. \tag{2.3}$$

The remaining decoding algorithms apply a further constant amplitude gain G_{SNR} to produce the scaled received symbol

$$\delta_i = G_{\mathrm{SNR}}\, \hat{r}_i = G_{\mathrm{AGC}}\, G_{\mathrm{SNR}}\, r_i. \tag{2.4}$$

The gain G_{SNR} represents scaling in inverse proportion to the output of an estimator for $\hat{\sigma^2}$, which can be considered equivalently as an estimator for the signal-to-noise ratio due to the relationship in equation (2.2). We use the latter terminology in referring to its use with the decoding algorithms. In the dissertation, we assume perfect estimation of the signal-to-noise ratio, resulting in

$$G_{\mathrm{SNR}} = \frac{2}{\hat{\sigma}^2}; \tag{2.5}$$

thus, from equations (2.1), (2.2), (2.4), and (2.5),

$$\delta_i = \frac{2\,(\sqrt{\mathcal{E}_c})}{\sigma^2}\, r_i. \tag{2.6}$$

for the receivers with the decoding algorithms using signal-to-noise ratio estimation.

The demodulator and scaling are optionally followed by a quantizer which provides a discrete approximation $\tilde{r}_i$ to each scaled received symbol δ_i as the input to the subsequent decoder. (The quantizer is present only for fixed-point decoding, otherwise the real-valued scaled received symbols are supplied directly to the decoder for high-precision floating-point decoding.) Finally, the output of the demodulator or

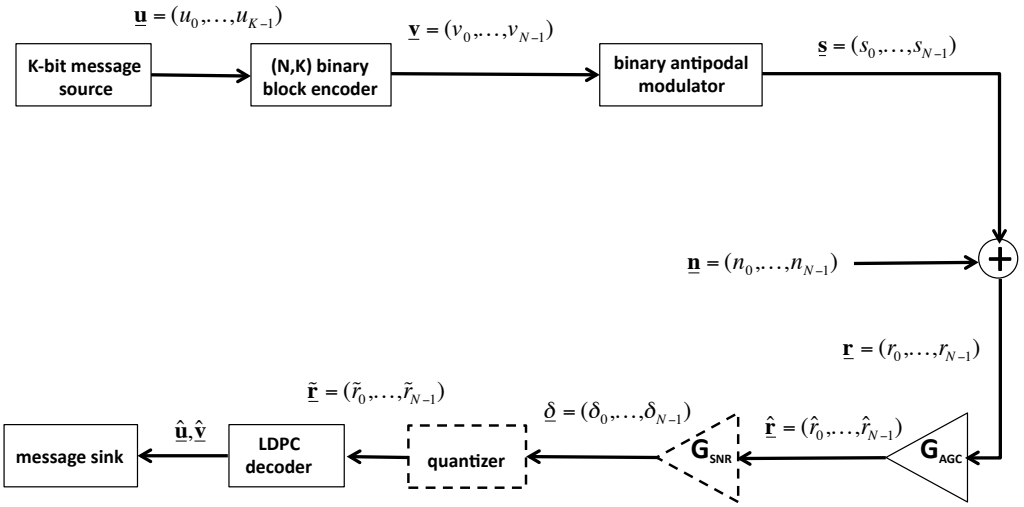

Figure 2.1: Coded system with an additive white Gaussian noise channel.

the quantizer is decoded to produce either a decoder failure or a detected code word $\hat{\underline{\mathbf{v}}}$ and a detected message $\hat{\underline{\mathbf{u}}}$. The decoder implements one of the decoding algorithms discussed in Chapters 4, 6, and 7.

2.2 Uniform quantizer

A uniform, mid-tread quantizer [41] is used in the receiver employing fixed-point decoding. It employs q bits of *precision* and a *resolution* of Δ; thus, the quantizer function is given by

$$\tilde{r}_i = \begin{cases} \max(N_{\max} - 1, \lfloor \frac{x}{\Delta} + .5 \rfloor) \times 2^{-q_f}, & \text{if } x \geq 0 \\ \min(-N_{\max}, \lceil \frac{x}{\Delta} - .5 \rceil) \times 2^{-q_f}, & \text{otherwise} \end{cases};$$

where $N_{\max} = 2^{q-1}$ and q_f is the number of bits to the right of the (implicit) decimal point in the fixed-point representation of the output. The *range* of the quantizer is $[-N_{\max}\ \Delta, (N_{\max} - 1)\ \Delta]$.

The fixed-point decoding algorithms considered in the dissertation use only three arithmetic operations: signed addition of operands, selection of the maximum operand, and selection of the minimum operand. The behavior of the algorithms it thus unaffected by the location of the decimal point in the fixed-point representation (if constants in the algorithm are scaled accordingly), and we assume without loss of generality that integer representation and integer operations are used (that is, $q_f = 0$). The resulting quantizer function is illustrated in Fig. 2.2.

The integer-valued outputs of the quantizer are restricted to the values of $\{-N_{\max}, \ldots, N_{\max} - 1\}$. The quantization intervals are symmetric about the input value r_i=0, except that the magnitude above which positive-valued inputs are clipped, $(N_{\max} - 1)\Delta$, is smaller than the magnitude above which negative-valued inputs are clipped, $(N_{\max})\Delta$. The set of quantizer outputs fit naturally with two's complement arithmetic [2], which is used in each of the results for fixed-point decoding in the dissertation.

2.3 Measures of system performance

Each decoding algorithm considered in the dissertation is an iterative algorithm with a pre-determined maximum number of iterations, $I_{\max}$. The number I of decoder iterations for a given received word may be fixed, or it may vary between received words depending on the random outcome of the channel noise process. It is thus a random variable in general.

For each algorithm, the probability of an undetected code-word error is much

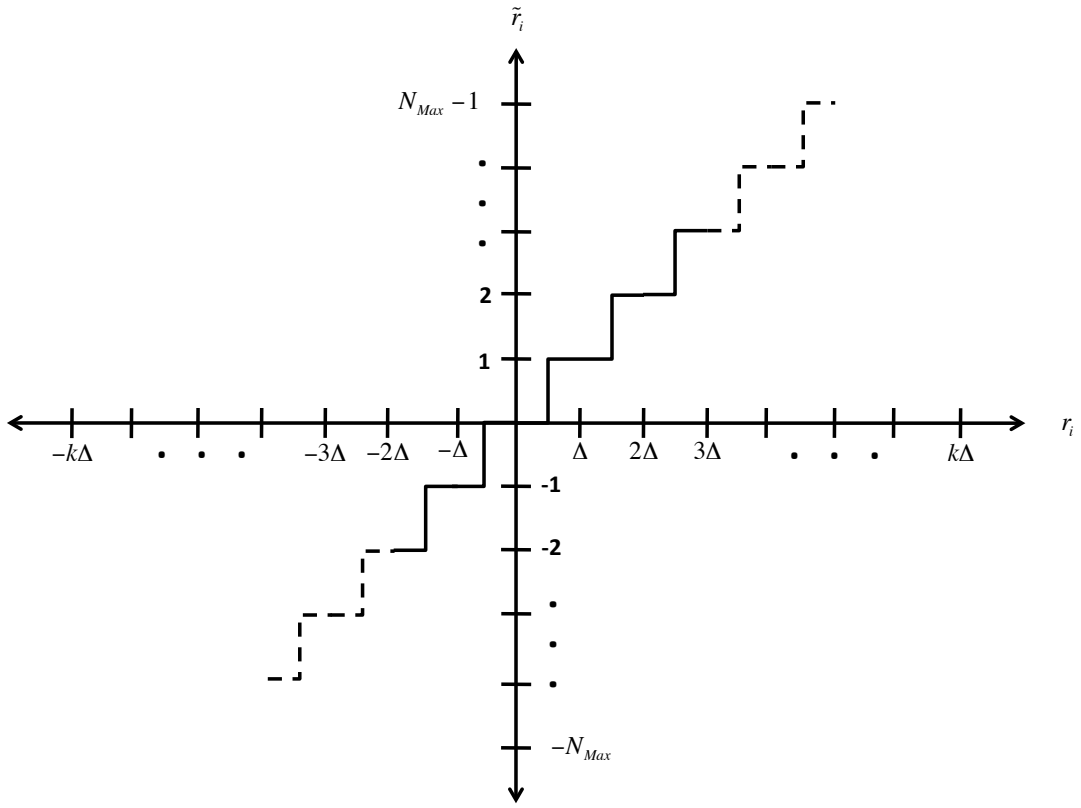

Figure 2.2: Uniform mid-tread quantizer.

smaller than the probability of a detected error (i.e, a decoder failure). The two measures of performance considered are thus the probability of not decoding correctly (which is approximately equal to the probability of a decoder failure) and the average (expected) number of iterations $\bar{I}$. In keeping with the terminology common in the literature on LDPC codes, we refer to the probability of not decoding correctly as the *probability of code-word error*. It is denoted by P_e.

All of the system performance results in Chapters 4-7 are obtained from Monte Carlo simulation of the entire system. Each numerical result in Chapter 8 is obtained using randomly generated received words as input to either the Storm-1 processor or an emulator provided as a development tool with the Storm-1 processor. Each

sample of the error probability or the average number of iterations is obtained from the larger of 10,000 trials and enough trials to result in 500 decoder failures. (No undetected code-word errors were observed in any of the simulations.)

Chapter 3

Low-Density Parity-Check Codes

3.1 Linear block codes

An (N, K) binary, linear block code is defined by a K-by-N binary *generator matrix* [1], $\mathbf{G}$, which maps each information word $\underline{\mathbf{u}}$ into a codeword $\underline{\mathbf{v}}$ by the linear transformation

$$\underline{\mathbf{v}} = \underline{\mathbf{u}}\mathbf{G}.$$

The computation required for encoding a code word of a linear code is a quadratic function of the block length N, for a given *code rate* K/N, unless the generator matrix contains some structure that can be exploited in the encoder. The computation required for the inverse mapping of a code word to its corresponding information word is also a quadratic function of N in general.

A *systematic code* [1] is a linear code in which each information word is mapped to a code word that includes the information word unaltered as part of the code word (conventionally represented as the final symbols of the code word); thus,

$$\underline{\mathbf{v}} = [p_0\ p_1 \ldots p_{N-K-1}\ u_0\ u_1 \ldots u_{K-1}]$$

where $(u_0, \ldots, u_{K-1})$ are the message bits and $(p_0, \ldots, p_{N-K-1})$ are *parity-check bits*. It follows that the generator matrix of a systematic code is given by

$$\mathbf{G} = [\mathbf{P}|\mathbf{I_K}],$$

where $\mathbf{P}$ is the K x $(N-K)$ *parity-generator submatrix* and $\mathbf{I_K}$ is the K x K identity matrix. A systematic code permits recovery of the information word from a code word directly with no computation.

The set of valid code words (i.e., the *code book*) of a binary, linear code forms a K-dimensional subspace under vector addition of the vector space of length-N binary vectors. The *dual subspace* [1] has dimension $(N-K)$, and an M-by-N matrix with binary row vectors that span the dual subspace is referred to as a *parity-check matrix* of the code. It follows that $\mathbf{v}$ is a code vector generated from $\mathbf{G}$ if and only if

$$\mathbf{vH^T} = \mathbf{0}.$$

Each row of $\mathbf{H}$ defines a *parity check* [1] that is satisfied by each valid code word, and a length-N binary vector is a valid code word if and only if it satisfies all M parity checks defined by $\mathbf{H}$. If $N-K$ is large, there are many distinct bases of the vector subspace spanned by the rows of $\mathbf{H}$; thus, the linear code has many distinct parity-check matrices. Any two codes differing only by a permutation of their code-symbol positions are said to be *equivalent*; their respective parity-check matrices differ

by the same column permutation.

3.2 LDPC codes

Low-density parity-check codes are a class of linear block codes that can achieve error performance approaching the Shannon capacity in an AWGN channel for very long block lengths [42, 43]. The codes were originally discovered in the early 1960's [44], but they were largely ignored until they were re-examined almost 20 years later when a graphical representation of the codes was developed [45]. A rapid growth of interest in LDPC codes began in the mid-1990's [46] when it was recognized that the belief-propagation iterative decoding algorithms that achieve near-maximum-likelihood decoding of the codes have computational requirements that can be met by modern digital micro-electronics.

Low-density parity-check codes are now one of two widely used modern classes of channel codes, the other being turbo codes [47]. They have several advantages over turbo codes. Good performance can be achieved with LDPC codes in a wide range of channels without the use of an interleaver [44]. For codes with a long block length or a high rate, LDPC codes provide better performance than turbo codes for a given decoding effort [48]. The error floor resulting with standard decoding algorithms for LDPC codes is typically much lower than the error floor for turbo codes of similar rate and block length [49]. The decoding algorithms for LDPC codes are also amenable to a much greater degree of parallelism than the decoding algorithms for turbo codes, providing a comparably greater flexibility in the trade-off between computing resources and decoding delay (e.g., see [50]).

An LDPC code is defined (and usually designed) in terms of a parity-check matrix of the code which has a low *density* (i.e,. a small fraction of non-zero entries).

Most of the properties of interest in the LDPC code can be characterized in terms of the properties of the corresponding low-density parity-check matrix $\mathbf{H}$. The number of non-zero entries per row of $\mathbf{H}$ is denoted w_{c_i}, $i = 1, \ldots, M$, and the number of non-zero entries per column of $\mathbf{H}$ is denoted w_{v_j}, $j = 1, \ldots, N$. The row weights $\{w_{c_i}\}$ and column weights $\{w_{v_j}\}$ of $\mathbf{H}$ can greatly affect the computational requirements of the decoding algorithms for the LDPC code (which are defined in terms of $\mathbf{H}$). If the column weight is the same for each column and the row weight is the same for each row, the LDPC code is characterized as *regular*. If the row weights are not identical or the column weights are not identical, the code is characterized as *irregular*. Encoder and decoder implementations are more complex in general for irregular LDPC codes than for regular LDPC codes.

The structure of the parity-check matrix of the code can be represented as a *Tanner graph* [45]. A Tanner graph is defined as a collection of two distinct types of nodes — *variable* nodes and *check* nodes — and edges, each of which connects a variable node and a check node. Each variable node represents one of the N code-symbol positions in the code word, while each check node represents the parity-check equation corresponding to one of the M rows of $\mathbf{H}$. A variable node is connected by a graph edge to a check node if the code-symbol position represented by the variable node participates in the parity-check equation represented by the check node (i.e., if a non-zero entry appears in the corresponding row and column of $\mathbf{H}$).

The representation of a parity-check matrix by its Tanner graph is illustrated in Fig. 3.1 for an example of a (7,3) code. Each row of the parity-check matrix represents a check node, c_j, while each column represents a variable node v_i. The code in the example is *row regular*, with a weight of three for each row in the parity-check matrix, while it is *column irregular*, with an average column weight of 12/7 and a maximum column weight of three in the parity-check matrix.

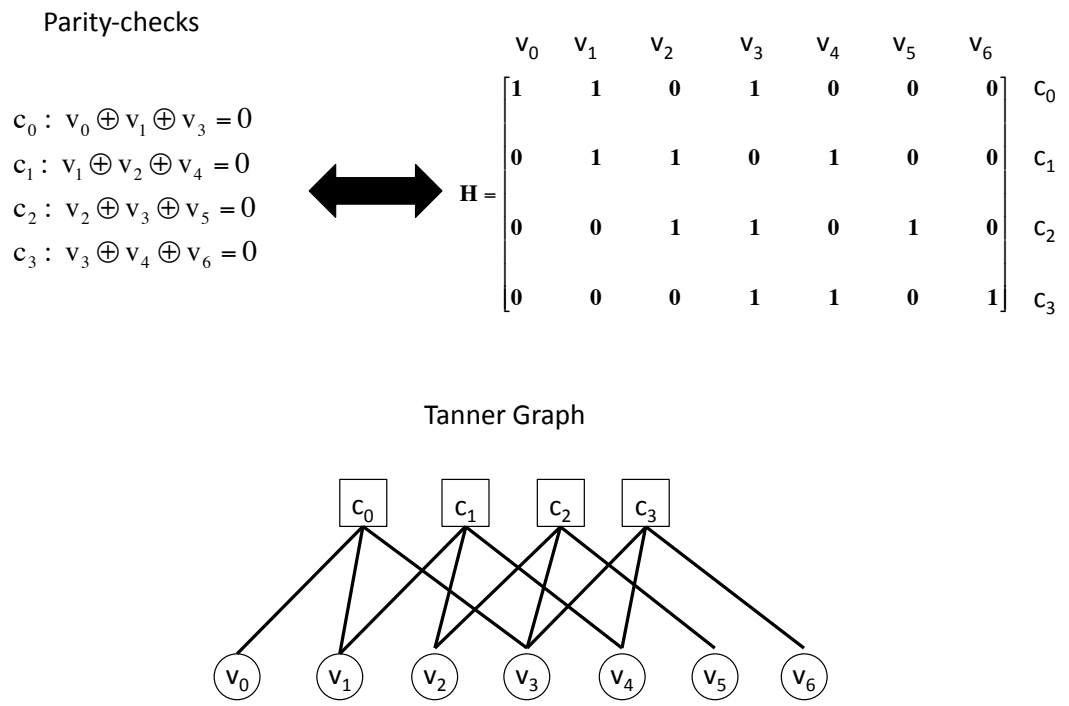

Figure 3.1: Tanner graph and check node representation for a parity-check matrix.

Most LDPC codes that result in good decoder performance have the property that no two code-symbol positions participate in any two parity checks in common in the parity-check matrix used in the decoding algorithm. This characteristic motivates the concepts of a *cycle* and the *girth* of a Tanner graph. A cycle is any closed path in the Tanner graph, originating from a variable node or a check node, that does not pass through an intermediate node twice. The girth of the graph is defined as the minimum cycle length of the graph. The characteristic noted above is equivalent to the condition that the girth of the Tanner graph is at least six. The Tanner graph for the parity-check matrix in Fig. 3.1 has a girth of six. Short cycles within the graph usually result in performance that is much poorer than maximum-likelihood decoding

for a decoding algorithm based on the corresponding parity-check matrix.

3.3 Architecture-aware LDPC codes

An LDPC code is described as "architecture aware" if its parity-check matrix used in decoding is designed to simplify an implementation of the decoder exploiting a high level of parallelism [20]. In particular, the structure of the parity-check matrix should lead to a marked reduction in the number of interconnects in a hardware (ASIC or FPGA) decoder or the occurrences of data-access conflicts and inter-processor communications in a parallel software implementation compared with a randomly constructed LDPC code of the same rate and block length with a parity-check matrix of the same density. Parity-check matrices with architecture-aware properties contain large sets of rows or columns that are mutually disjoint (or nearly so) in the locations of their non-zero entries. Two classes of AA-LDPC codes are used as examples in the dissertation: *quasi-cyclic LDPC (QC-LDPC) codes* [1] and the non-QC LDPC codes defined in the DVB-S2 standard [8].

3.3.1 Quasi-cyclic LDPC codes

An (mn_0, mk_0) linear block code is *quasi-cyclic* with *shifting constraint* $n_0 > 1$ if every n_0-step right circular shift of a code word results in another code word [1]. An appropriate re-ordering of the code-symbol positions results in an equivalent linear block code in which the generator matrix (and a corresponding parity-check matrix) are composed of (square) m-by-m circulant submatrices and all-zeros submatrices [1]. Thus it is possible to define a QC-LDPC code by constructing a low-density parity-check matrix composed of circulant submatrices and all-zeros submatrices of the same size.

A particularly appealing construction of the parity-check matrix for a QC-LDPC code is through the use of *circulant permutation submatrices* $\mathrm{I}_{\mathbf{m}}^{(\mathbf{j})}$, the j-step right circular shift of the m-by-m identity matrix, for various values of j. (Quasi-cyclic LDPC codes with circulant permutation submatrices are also referred to as "block type" LDPC codes [52].) For example,

$$\mathrm{I}_{\mathbf{4}}^{(\mathbf{2})} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix},$$

is the 4-by-4 circulant permutation matrix formed from the two-step right circular shift of the 4-by-4 identity matrix.

The parity-check matrix is constructed so that no two identical circulant permutation sub matrices appear in the same row block or column block, thus preventing the occurrence of cycles of length four in the Tanner graph. The placement of the submatrices within an $(m(n_0 - k_0))$-by-(mn_0) parity-check matrix is determined by an $(n_0 - k_0)$-by-n_0 *base matrix* [7]. Each non-zero entry in the base matrix indicates the location of an m-by-m circulant permutation submatrix, and each zero entry in the base matrix indicates the position of an m-by-m all-zeros submatrix. The density of the resulting parity-check matrix is $1/m$ times the density of the base matrix. The row-weight distribution and the column-weight distribution of the parity-check matrix also depend only on its base matrix.

If the entries of the base matrix are replaced by entries with values that designate the shift of their corresponding m-by-m submatrices, it is referred to as a *base model matrix* [7].

Quasi-cyclic LDPC codes can be encoded efficiently using a shift-register circuit [51], and QC-LDPC codes with parity-check matrices constructed from circulant permutation submatrices are amenable to particularly efficient shift-register-based encoding [52]. The latter QC-LDPC codes also have the characteristics of architecture-aware codes: no two rows within an m-row block of submatrices has any non-zero positions in common, and no two columns within an m-column block of submatrices has any non-zero positions in common. Both properties can be exploited to achieve a high degree of parallelism in various decoding algorithms. The first of the two properties can be exploited with any of the decoders considered in Chapter 6–8;it is used explicitly in the decoders in Chapter 8.

3.4 Examples of AA-LDPC codes

Three types of AA-LDPC code will be used in the examples in the following chapters. Two of them, the Sridhara-Fuja-Tanner (SFT) codes [53] and the WiMAX codes [7], are examples of QC-LDPC codes with parity-check matrices composed of circulant permutation submatrices. The third type, the DVB-S2 codes [8], are AA-LDPC codes that are not quasi-cyclic.

3.4.1 Sridhara-Fuja-Tanner LDPC Codes

An SFT LDPC code is a type of QC-LDPC code that is composed of circulant permutation matrices. It is constructed by first choosing an m x m permutation circulant matrix κ other than the identity matrix for some odd prime number m. The multiplicative group in the set of integers modulo m is used to place circulant submatrices inside a parity-check matrix, where each submatrix is an appropriate power of κ. Specifically, the parity-check matrix for the SFT code is given by

$$\mathbf{H} = \begin{bmatrix} \kappa & \kappa^{a} & \kappa^{a^2} & \cdots & \kappa^{a^{k-1}} \\ \kappa^{b} & \kappa^{ab} & \kappa^{a^2 b} & \cdots & \kappa^{a^{k-1} b} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \kappa^{b^{j-1}} & \kappa^{ab^{j-1}} & \kappa^{a^2 b^{j-1}} & \cdots & \kappa^{a^{k-1} b^{j-1}} \end{bmatrix}$$

where a and b are nonzero elements of GF(m) such that ord(a) = n_0 and ord(b) = $n_0 - k_0$. The exponent $b^s a^t$, where $0 \leq s < n_0 - k_0$ and $0 \leq t < n_0$, is calculated in the field GF(m). The $(m(n_0 - k_0))$-by-(mn_0) parity-check matrix does not necessarily have full row rank; the resulting linear code has a rate of k_0/n_0 or greater.

The LDPC codes constructed in this manner are quasi-cyclic and regular with shifting constraint n_0, row weight $w_c = n_0$, and column weight $w_v = n_0 - k_0$ [53]. An SFT code can be constructed for various choices of the column weight w_v and the row weight w_c. The construction technique can be generalized to non-prime m [53]. For short-to-moderate block sizes, SFT codes perform as well as randomly constructed regular LDPC codes when iterative message-passing decoding is used in an AWGN channel [53].

3.4.2 WiMAX LDPC codes

The LDPC codes specified in IEEE Std. 802.16e [7] (the "WiMAX" standard) are systematic QC-LDPC codes with parity-check matrices constructed from circulant permutation submatrices. The general form of the parity-check matrices is given by

$$\mathbf{H}_{R \times S} = [\mathbf{H1}_{R \times (S-R)} | \mathbf{H2}_{R \times R}]$$

$$= \left(\begin{array}{ccc|cccccc} \mathbf{P}_{0,0} & \cdots & \mathbf{P}_{0,S-R-1} & \mathbf{P}_{0,S-R} & \mathbf{I} & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ \mathbf{P}_{1,0} & \cdots & \mathbf{P}_{1,S-R-1} & \vdots & \mathbf{I} & \mathbf{I} & & & \mathbf{0} \\ \vdots & \cdots & \vdots & \vdots & \mathbf{0} & \mathbf{I} & \ddots & & \vdots \\ \vdots & \cdots & \vdots & \vdots & \ddots & \ddots & \ddots & \mathbf{I} & \mathbf{0} \\ \mathbf{P}_{R-2,0} & \cdots & \mathbf{P}_{R-2,S-R-1} & \vdots & & \ddots & & \mathbf{I} & \mathbf{I} \\ \mathbf{P}_{R-1,0} & \cdots & \mathbf{P}_{R-1,S-R-1} & \mathbf{P}_{R-1,S-R} & \mathbf{0} & & \cdots & \mathbf{0} & \mathbf{I} \end{array}\right) \quad (3.1)$$

where each submatrix $\mathbf{P_{i,j}}$ is a z-by-z matrix that is either the all-zeros matrix or a circulant permutation matrix and I is the z-by-z identity submatrix.

A total of 76 LDPC codes are specified in the WiMAX standard; their parameters are listed in Table 3.1. They include four choices of the code rate – 1/2, 2/3, 3/4, and 5/6 – and 19 block lengths for each code rate, ranging from 576 bits to 2304 bits. The parameter S in equation (3.1) is equal to 24 for all the codes. The parameter R is equal to three, six, nine, and twelve for the codes of rates 5/6, 3/4, 2/3, and 1/2, respectively; it does not depend on the block length of the code. The block length of the code is determined by the size z of the circulant permutation submatrices in the parity-check matrix, which ranges from $z = 24$ for the codes of block length 576 to $z = 96$ for the codes of block length 2304.

Each WiMAX code has a parity-check matrix derived from a base model matrix consistent with the form of equation (3.1). All the codes of rate 1/2 have the same base matrix with the shift of each circulant permutation submatrix depending on the block length of the code. Similarly, all the codes of rate 5/6 have the same base matrix. For the codes of rate 2/3, there are two choices of the base matrix; the resulting codes are designated as "rate-2/3 A codes" and "rate-2/3 B codes". Similarly, there are two base matrices for the rate-3/4 codes, and the resulting codes

are also distinguished by the "A" and "B" suffixes.

The WiMAX codes are irregular. The weight distributions for each of the six base matrices are shown in Table 3.2. For example, each rate-1/2 code has column weights of two, three, and six. Of the 24 groups of z columns in its parity-check matrix, eleven have a weight of two in each column of the group, eight groups have a column weight of three, and five groups have a column weight of six.

3.4.3 DVB-S2 LDPC codes

The DVB-S2 standard [8] specifies systematic LDPC codes that are a form of irregular repeat-accumulate (IRA) code [54]; thus, the codes are sometimes referred to as LDPC-IRA codes [55]. The DVB-S2 standard offers two distinct LDPC code lengths: *normal frame* codes of block length 64,800, and *short frame* codes with block length 16,200. Each is the inner code in a serial concatenation with a high-rate outer BCH code [1]. There are eleven normal-frame LDPC codes with rates ranging from 1/4 to 9/10 and ten short-frame LDPC codes with rates between 1/5 and 8/9.

The parity-check matrix $\mathbf{H}$ for each DVB-S2 code of the form

$$\mathbf{H}_{(N-K)\times N} = [\mathbf{A}_{(N-K)\times K} | \mathbf{B}_{(N-K)\times(N-K)}]$$

$$= \left(\begin{array}{ccc|cccccc} a_{0,0} & \cdots & a_{0,K-1} & 1 & 0 & \cdots & \cdots & 0 & 0 \\ a_{1,0} & \cdots & a_{1,K-1} & 1 & 1 & 0 & & 0 & 0 \\ \vdots & \cdots & \vdots & 0 & 1 & \ddots & & \vdots & \\ \vdots & \cdots & \vdots & \vdots & \ddots & \ddots & 1 & 0 & 0 \\ a_{N-K-2,0} & \cdots & a_{N-K-2,K-1} & \vdots & \ddots & & 1 & 1 & 0 \\ a_{N-K-1,0} & \cdots & a_{N-K-1,K-1} & 0 & & \cdots & 0 & 1 & 1 \end{array}\right). \quad (3.2)$$

The left submatrix **A** consists of z-by-z submatrices (with z as large as 360), each of which is either an all-zeros matrix, a circulant permutation submatrix, or the sum of two distinct circulant permutation submatrices [56]. The right submatrix **B** is a staircase lower triangular matrix. however; The right submatrix **B** imparts the repeat-accumulate properties of the code and results in a simple form for the systematic encoder. The code is not quasi-cyclic.

The form of the left submatrix **A** limits the storage required for encoding and decoding. The structure of **H** differs from the form of a parity-check matrix of circulant permutation submatrices. Parallelism can still be exploited in the decoding algorithm as if the parity-check matrix consisted only of such submatrices, however, though at the cost of a slight degradation in performance [56].

Code	**Block length (N)**	**Circulant size (z)**	**Dimension (K)**			
			rate	**rate**	**rate**	**rate**
			1/2	2/3	3/4	5/6
1	576	24	288	384	432	480
2	672	28	336	448	504	560
3	768	32	384	512	576	640
4	864	36	432	576	648	720
5	960	40	480	640	720	800
6	1056	44	528	704	792	880
7	1152	48	576	768	864	960
8	1248	52	624	832	936	1040
9	1344	56	672	896	1008	1120
10	1440	60	720	960	1080	1200
11	1536	64	768	1024	1152	1280
12	1632	68	816	1088	1224	1360
13	1728	72	864	1152	1296	1440
14	1824	76	912	1216	1368	1520
15	1920	80	960	1280	1440	1600
16	2016	84	1008	1344	1512	1680
17	2112	88	1056	1408	1584	1760
18	2208	92	1104	1472	1656	1840
19	2304	96	1152	1536	1728	1920

Table 3.1: Parameters of the WiMAX codes.

Code	w_v (column weight)	w_c (row weight)
1/2	[2,3,6] = $\{11/24, 1/3, 5/24\}$	[6,7] =$\{2/3, 1/3\}$
$2/3A$	[2,3,6] = $\{7/24, 1/2, 5/24\}$	[10] =$\{1\}$
$2/3B$	[2,3,4] = $\{7/24, 1/24, 2/3\}$	[10,11] =$\{7/8, 1/8\}$
$3/4A$	[2,3,4] = $\{5/24, 1/24, 3/4\}$	[14,15] =$\{5/6, 1/6\}$
$3/4B$	[2,3,6] = $\{5/24, 1/2, 7/24\}$	[14,15] =$\{1/3, 2/3\}$
5/6	[2,3,4] = $\{3/24, 5/12, 11/24\}$	[20] =$\{1\}$

Table 3.2: WiMAX code weight distributions.

Chapter 4

Benchmark Decoding Algorithms for LDPC Codes

4.1 Iterative message-passing algorithms

Near-maximum-likelihood decoding of LDPC codes in the communication system of Section 2.1 can be achieved with any of several iterative message-passing algorithms based on belief propagation that are defined in terms of a parity-check matrix of the code, or equivalently, its Tanner graph. A code word $\underline{\mathbf{v}}$ transmitted over the channel results in the received word $\underline{\mathbf{r}}$, which is used by the receiver to determine the *intrinsic value* for each code symbol (variable node) v_i given by estimating its channel-symbol log-likelihood ratio (LLR)

$$\delta_{v_i} = \log\left(\frac{\Pr(r_i \mid v_i = 0)}{\Pr(r_i \mid v_i = 1}\right) = 2\,\frac{\sqrt{\mathcal{E}_c}}{\sigma^2} r_i.$$

In order to determine the intrinsic value, the receiver employs normalization of the received symbols with respect to both the amplitude of the received signal and its

signal-to-noise ratio. Under the assumption of perfect estimation, the resulting scaled received symbol δ_i is exactly the LLR of r_i.

Each algorithm can be described in terms of exchanges of two types of message [57]. The *variable message* denoted $\mu_{v_i \to c_j}$ represents information provided by variable node v_i to check node $c_j \in \mathcal{N}(v_i)$, where $\mathcal{N}(v_i)$ represents the set of check nodes connected with variable node v_i in the Tanner graph. (In other words, the (i, j)th entry in the parity-check matrix is non-zero.) Similarly, the *check message* denoted $\mu_{c_j \to v_i}$ represents information provided by check node c_j to variable node $v_i \in \mathcal{N}(c_j)$, where $\mathcal{N}(c_j)$ represents the set of variable nodes connected with check node c_j in the Tanner graph. The variable message $\mu_{v_i \to c_j}$ is proportional to the current approximated LLR δ_i for code symbol v_i given the most recent check messages received by the variable node v_i from all check nodes in $\mathcal{N}(v_i)/\{c_j\}$. Similarly, check message $\mu_{c_j \to v_i}$ is proportional to the current approximated LLR for code symbol v_i given the most recent variable messages received by the check node c_j from all variable nodes in $\mathcal{N}(c_j)/\{v_i\}$.

The intrinsic values are used as initial values stored at their corresponding variable nodes. Iterations of message exchanges between variable nodes and their connected checks nodes are performed, and each iteration results in an update of the approximated LLRs for the code symbols (referred to as *posterior values*). The message exchanges in each iteration are referred to as the *message-passing phase* for the iteration, and the order in which the messages are exchanged in an iteration is the *schedule* used by the algorithm. At the end of the message-passing phase for each iteration, the *parity-check phase* occurs in which tentative hard decisions are made for each code symbol based on its current posterior value. A test is performed to determine if the vector of hard-decision values satisfies all the parity checks (and thus represents a valid code word). If it does, the detected code word is provided as the

output and the algorithm terminates. Otherwise, another iteration is executed unless the maximum number of iterations has already been executed. In the latter case, a decoder failure is declared and the algorithm terminates.

The SPA and the TDMP algorithm use different message-passing schedules which are described in Sections 4.1.1 and 4.1.2. The difference in the schedules and the presence of cycles in the Tanner graph result in different outcomes in general for the two algorithms [22]. The presence of cycles also results in non-maximum-likelihood decoding with both algorithms.

4.1.1 Sum-Product Algorithm

The SPA employs a two-stage message passing schedule in which the variable messages for all N variable nodes are sent to their connected check nodes, then the check messages for all M check nodes are sent to their connected variable nodes. The algorithm is described in the steps below, where $\text{sgn}(x)$ represents the sign of x.

1. For all variable nodes v_i, initialize $\mu_{v_i \to c} = \delta_{v_i}$ for all $c \in \mathcal{N}(v_i)$.

2. Let $\alpha_{v_i \to c_j} = \text{sgn}(\mu_{v_i \to c_j})$ and $\beta_{v_i \to c_j} = |\mu_{v_i \to c_j}|$. For all check nodes, set

$$\mu_{c_j \to v} = \left(\prod_{s \in \mathcal{N}(c_j), s \neq v} \alpha_{s \to c_j} \right) \cdot \phi \left(\sum_{s \in \mathcal{N}(c_j), s \neq v} \phi(\beta_{s \to c_j}) \right)$$

for all $v \in \mathcal{N}(c_j)$, where the function ϕ is defined by

$$\phi(x) = -\log \left[\tanh \left(\frac{x}{2} \right) \right] = \log \left(\frac{e^x + 1}{e^x - 1} \right). \tag{4.1}$$

3. For all variable nodes, set

$$\mu_{v_i \to c} = \delta_{v_i} + \sum_{w \in \mathcal{N}(v_i), w \neq c} \mu_{w \to v_i} \text{ , for all } c \in \mathcal{N}(v_i).$$

4. Make hard decisions for each code bit

$$v_j = \frac{1}{2}\left(\text{sgn}\left(\delta_{v_j} + \sum_{s \in \mathcal{N}(v_j)} \mu_{s \to v_j}\right) + 1\right)$$

for $j = 1,\ldots,N$.

5. Steps 2–4 represents one iteration of the SPA. If all parity-checks are satisfied from the hard decisions in step 4, then decoding is complete and the information word is recovered by inverse mapping. Otherwise, if the decoder has executed the maximum number of iterations, the decoder fails. If neither is true, another iteration of the algorithm is performed starting with step 2.

All variable messages can be passed concurrently and all check messages can be passed concurrently in the message-passing phase of the SPA without altering the logic of the algorithm. Similarly, all parity checks can be performed concurrently in the parity-check phase. Consequently, the SPA is amenable to any degree of parallelism of either stage in its two-stage message-passing schedule as well as its parity-check phase. This characteristic has been exploited in numerous high-data-rate hardware-based decoders.

4.1.2 Turbo-Decoding Message-Passing Algorithm

The TDMP algorithm employs a single-stage message-passing schedule (i.e., a "merged" message-passing schedule), in contrast with the two-stage message-passing

schedule of the SPA. Specifically, message passing in the TDMP algorithm is scheduled in check-node order. For each check node in turn, the variable messages into the check node are exchanged, followed by the check messages from the check node. Posterior updates associated with check messages from a given check node thus benefit from the improvements in the posterior values from earlier message exchanges in the iteration, in contrast with the SPA.

One can view the TDMP algorithm as an algorithm executed in turn for each row of the parity-check matrix from the top row to the bottom row; hence it is said to employ "row-wise scheduling". Row-wise LDPC decoding algorithms are also referred to variously by the terms "staggered decoding" [58], "layered decoding" [21], and "Gauss-Siedel decoding" [59]. (Another layered-decoding algorithm, the shuffled belief-propagation algorithm [50], uses a column-wise message-passing schedule.)

The TDMP algorithm is described in the steps below. The notation and terminology are taken from [22]. The terminology is motivated by a turbo-decoder interpretation of the posterior update step (which is also the basis for the name of the algorithm).

Let the vector $\underline{\boldsymbol{\lambda}}^{\boldsymbol{i}} = [\lambda_1^i, \ldots, \lambda_{w_{c_i}}^i]$ represent the *extrinsic messages* that correspond to the non-zero entries in row i for each row of the parity-check matrix $\mathbf{H}$, where w_{c_i} is the weight of row i. The notation I_i denotes the list of the column positions of non-zero entries in row i in $\mathbf{H}$. The subset of the posterior values corresponding to the non-zero column positions of row i are denoted $\mathbf{w}(I_i)$.

1. Initialize extrinsic messages $\underline{\boldsymbol{\lambda}}^{\boldsymbol{i}} = \underline{\mathbf{0}}$, for $i = 1, \ldots, M$. Also, initialize the posterior values $\mathbf{w} = [\delta_1, \ldots, \delta_N]$ with the intrinsic values derived from the received word.

2. Read the extrinsic messages $\underline{\boldsymbol{\lambda}}^{\boldsymbol{i}}$ and the posterior values $\mathbf{w}(I_i)$ for row i of the

parity-check matrix.

3. Subtract $\underline{\boldsymbol{\lambda}}^{\boldsymbol{i}}$ from $\mathbf{w}(I_i)$ to generate *prior messages* $\underline{\boldsymbol{\rho}} = [\rho_1, \ldots, \rho_{w_{c_i}}] = \mathbf{w}(I_i)$ - $\underline{\boldsymbol{\lambda}}^{\boldsymbol{i}}$.

4. Calculate extrinsic messages based on the parity-check equation for row i. Define $\underline{\boldsymbol{\alpha}} = [\alpha_1, \ldots, \alpha_{w_{c_i}}]$ and $\underline{\boldsymbol{\beta}} = [\beta_1, \ldots, \beta_{w_{c_i}}]$, where $\alpha_j = \text{sgn}[\rho_j]$ and $\beta_j = |\rho_j|$. Set
$$\lambda_j^i = \left(\prod_{k=1, k \neq j}^{w_{c_i}} \alpha_k \right) \cdot \phi \left(\sum_{k=1, k \neq j}^{w_{c_i}} \phi(\beta_k) \right)$$
for $j = 1, \ldots, w_{c_i}$,

5. Update the posterior values for the bit positions of I_i as
$\mathbf{w}(I_i) = \boldsymbol{\rho} + \underline{\boldsymbol{\lambda}}^{\boldsymbol{i}}$.

6. Steps 2–5 represent a decoding subiteration for one row of $\mathbf{H}$. Repeat these steps for each row.

Steps 2–6 represent the message-passing phase of one iteration of the TDMP algorithm. It is followed by the parity-check phase of the iteration, using a hard decision for each v_i based upon the sign of γ_i, that is identical to the parity-check phase of the SPA.

The degree of parallelism that is possible with the TDMP algorithm depends on the structure of the parity-check matrix used for decoding. If the parity-check matrix consists of z-by-z circulant permutation submatrices (such as the parity-check matrices for the SFT and WiMAX codes), steps 2–5 of the TDMP algorithm can be executed in parallel for the z rows in any single row block of submatrices without altering the logic of the algorithm. Parallel execution for a larger number of rows results in a modification of the algorithm in general. If the parity-check matrix does

not have the desired structure (such as the parity-check matrices for the DVB-S2 code), any parallelism in the execution of the TDMP algorithm results in an alteration of its logic. The alteration typically degrades the performance of the algorithm, but the degradation is not necessarily excessive [56].

4.2 Extrinsic update approximation

The function $\phi(x)$ defined by equation (4.1) appears in both the SPA and the TDMP algorithm, but it is costly to implement in a practical decoder. It is typically implemented as a look-up table; however, the implementation is prone to quantization noise that results from the nonlinearity of $\phi(x)$, it requires memory, and each use of it incurs a memory-access delay. Increasing the precision of the look-up table results in a substantial increase in the computation required by the decoder [22].

An alternative to the look-up table is an approximating function to $\phi(x)$ that is designed for computational simplicity and reasonable accuracy. The *max-quartet (MQ)* function [22] replaces the extrinsic message in step 4 of the TDMP algorithm with the alternative extrinsic message

$$\lambda_j^i = Q(\ldots(Q(Q(\rho_1, \rho_2), \rho_3), \ldots), \rho_{w_{c_i}}) \tag{4.2}$$

where

$$Q(x, y) = \max(x, y) + \max(C_Q - \frac{|x - y|}{4}, 0) - \max(x + y, 0) - \max(C_Q - \frac{|x + y|}{4}, 0).$$

A value of $C_Q = 0.625$ is used in [20], but in general an optimal value of C_Q must be determined from simulation for the particular LDPC code and the operating condition of interest.

Evaluation of the message can be implemented in fixed-point arithmetic if the constant C_Q is of radix two. It provides a close approximation to $\phi(x)$ while only requiring addition, subtraction, the max and min functions, and right shifts for division by a power of two. The approximation in equation (4.2) can also be used for the check message in the SPA. The SPA and the TDMP algorithm using the MQ approximation are designated as the MQ-SPA and the MQ-TDMP algorithm, respectively. As with the SPA and the TDMP algorithm, estimation of the signal-to-noise ratio is required for the MQ-SPA and the MQ-TDMP algorithm.

A simpler approximation to $\phi(x)$ is given by the *offset-min-sum (OMS)* function which replaces the extrinsic message in step 4 of the TDMP algorithm with the alternative extrinsic message

$$\lambda_j^i = \left(\prod_{k=1,k\neq j}^{w_{c_i}} \alpha_k \right) \cdot \max \left(\min_{1\leq k\leq w_{c_i}, k\neq j} \beta_k - \eta, 0 \right).$$

The value of the constant *offset* η must be optimized through simulation for the particular LDPC code and the operating condition of interest. The approximation can also be used in the SPA. The SPA and the TDMP algorithm using the OMS approximation are designated as the OMS-SPA and the OMS-TDMP algorithm, respectively. Unlike the SPA, the TDMP algorithm, the MS-SPA, and the MS-TDMP algorithm, the OMS-SPA and the OMS-TDMP algorithm do not require an estimate of the signal-to-noise ratio. Consequently, the systems using the OMS-SPA and the OMS-TDMP algorithm are modeled as using the perfect amplitude normalization in equation(2.1), but not the normalization with respect to the signal-to-noise ratio. The OMS-based algorithms can also provide better performance than their MQ-based counterparts at a sufficiently high signal-to-noise ratio for a given set of values of the algorithms' parameters [14].

4.3 Relative performance of the benchmark algorithms

The floating-point performance is considered in this section for some of the benchmark algorithms defined in the previous sections. In the results illustrated in Figs. 4.1-4.4, the performance of the SPA and the TDMP algorithm (using the function $\phi(x)$ with each) is compared for different constraints on the decoding delay. It is assumed that perfect estimates of the signal-to-noise ratio are available at the receiver for both algorithms. The SPA and the TDMP algorithm result in a similar computational burden per iteration; thus, the number of iterations serves as a fair basis of comparison of the decoding delay incurred with the algorithms if they are implemented using serial processing. Two SFT codes are used in the examples: a (305,124) code and a (2105, 844) code. They are referred to in the subsequent discussion as "the short code," and "the long code," respectively.

The rates of the short code and the long code are 0.4065 and 0.4009, respectively; thus they have nearly the same rate but different block lengths. The parity-check matrix for the short code includes circulant submatrices of size 61; the submatrix size for the long code is 421. Both parity-check matrices are regular and have row weights of three and column weights of five.

The performance of both the SPA and the TDMP algorithm with the short code is illustrated in Fig. 4.1. The figure shows the probability of code-word error as a function of the signal-to-noise ratio for each algorithm with a maximum of one, five, and 50 iterations. The two algorithms converge to nearly the same error probability as the maximum number of iterations increases, and the performance of the two is very close if the maximum is 50 iterations. As noted in [22], the TDMP algorithm converges to the limiting performance more quickly than the SPA.

The performance of the two decoding algorithms with the short code is shown in Fig. 4.2 for circumstances in which the decoder operates under a hard real-time constraint specified by the maximum number of decoder iterations that can be used to detect any block of data. Specifically, the performance is shown as the signal-to-noise ratio required to achieve a target probability of code-word error as a function of the maximum number of decoder iterations. Results are shown for two values of the target probability of code-word error: 10^{-2} and 10^{-3}. (Results for a target probability of code-word error of 10^{-1} are not shown; they yield similar comparisons to the results that are shown.)

Suppose the target probability of block error is 10^{-2} for a system using the short code. If the maximum number of iterations per data block is large (e.g., 50), both the SPA and the TDMP algorithm approach their limiting performance. A signal-to-noise ratio of 2.5 dB is required to achieve the target performance with the TDMP algorithm; the performance is 0.1 dB poorer if the SPA is used instead. If a more severe real-time constraint is placed on the decoder, however, the TDMP algorithm results in much better performance than the SPA. The signal-to-noise ratio required to achieve the target performance is 0.3 dB lower for the TDMP algorithm than the SPA if the decoder is limited to ten iterations per data block. The difference in performance increases to 0.8 dB if the maximum number of iterations is five, and it is 1.8 dB with a limit of two iterations.

The relative performance of the two decoders as a function of the maximum number of iterations remains the same if the target probability of code-word error is decreased to 10^{-3} for the system with the short code. The difference in performance ranges from nearly 2 dB if there is a very stringent real-time limit to only a small difference if a large number of iterations is allowed. Note that for an iteration limit of six or less, use of the TDMP algorithm allows the system to achieve a block error

probability of 10^{-3} with a lower signal-to-noise ratio than is required to achieve a block error probability of 10^{-2} if the SPA is used.

The difference in the performance of the two decoders under a hard real-time decoding constraint depends on the code that is employed. For either the short or long code, the signal-to-noise ratio that results in a probability of code-word error of 10^{-2} is approximately the same with either algorithm if up to 50 iterations are allowed per block. The TDMP algorithm results in better performance than the SPA for each code if a more stringent constraint is imposed, however, and the difference in the performance of the two algorithms increases with increasing code block length for a given constraint.

This is illustrated by comparing the results in Figs. 4.2 and 4.3, which show the performance for the short code and the long code, respectively, as a function of the maximum number of decoder iterations per received word. If a maximum of ten iterations is allowed, the TDMP algorithm results in a performance improvement of 0.3 dB relative to the SPA for the short code. The corresponding improvement is 0.6 dB for the long code. If the maximum number of iterations is five, the difference in performance is 0.8 dB and 1.4 dB for the short code and long code, respectively. The respective differences are 1.9 dB and 2.2 dB if a maximum of two iterations is allowed per received word. Similar results arise in the comparisons if a target probability of code-word error of 10^{-3} is considered.

The performance of the SPA and the TDMP algorithm is shown in Fig. 4.4 for circumstances in which the decoder operates under a soft real-time constraint that is specified by the average number of decoder iterations that can be used to detect a code word. (Note that the average number of decoder iterations may be much smaller than the maximum number of iterations, since many received words are decoded in fewer iterations than the maximum.) The signal-to-noise ratio required to achieve a

target probability of code-word error is shown as a function of the average number of decoder iterations with the short code. Once again, results are shown for two values of the target probability of code-word error: 10^{-2} and 10^{-3}. (Similar results for a target probability of code-word error of 10^{-1} are not shown.)

Suppose the target probability of code-word error is 10^{-2}. If the *maximum* number of iterations is large, both the SPA and the TDMP algorithm result in performance close to their (similar) respective limiting values. For example, signal-to-noise ratios of approximately 2.5 dB and 2.6 dB are required for the TDMP algorithm and the SPA, respectively, to achieve the target performance with a maximum of 50 iterations. The corresponding *average* number of iterations differs for the two algorithms in that instance, however. The TDMP algorithm requires an average of about 3.9 iterations per received word if the maximum is 50, whereas the SPA requires and average of about 6.1 iterations.

A target probability of code-word error of 10^{-2} is achieved with the TDMP algorithm with a signal-to-noise ratio that is 1.5 dB lower than the signal-to-noise ratio required with the SPA, if each decoder is limited to an average of 2.5 iterations. The same difference arises if the average is two iterations. The difference in performance increases to 1.7 dB if the average number of iterations is limited to 1.5, and it is 1.8 dB if only a single iteration is allowed for each received word. Similar results are seen for a target probability of code-word error of 10^{-3}, except that the TDMP algorithm and SPA results in averages of 2.7 and 4.3 iterations, respectively, if the maximum number of iterations is 50.

The performance of the MQ-TDMP algorithm and the OMS-TDMP algorithm is compared in Fig. 4.5 for the (2304,1152) WiMAX code and a maximum of either 20 iterations or 50 iterations. The parameters in both algorithms are optimized for the performance at a signal-to-noise ratio of 2 dB, and it is assumed that a perfect

estimate of the signal-to-noise ratio is available at the receiver in the system using the MQ-TDMP algorithm.

For a maximum of 50 decoding iterations and a probability of code-word error 10^{-2} and 10^{-3}, the MQ-TDMP algorithm outperforms the OMS-TDMP algorithm by .08 dB and .05 dB, respectively. Similar relative performance of the two algorithms occurs if the maximum number of decoding iterations is reduced to 20. As noted earlier, the OMS-TDMP algorithm can outperform the MQ-TDMP algorithm at a sufficiently high signal-to-noise ratio. In this example, the OMS-TDMP algorithm outperforms the MQ-TDMP algorithm if the signal-to-noise ratio is greater than 1.93 dB is the maximum number of iterations is 50. Similar relative performance is observed at high signal-to-noise ratios if the maximum number of iterations is 20. For either value of I_{max}, over the range of performance shown in Fig. 4.5, the OMS-TDMP algorithm requires an average of up to 5% more iterations to achieve a given error probability than the MQ-TDMP algorithm. Although not shown in Fig. 4.5, the MQ-TDMP algorithm results in performance that is almost identical to the performance with the standard TDMP algorithm utilizing the function $\phi(x)$, which is consistent with previous results [20].

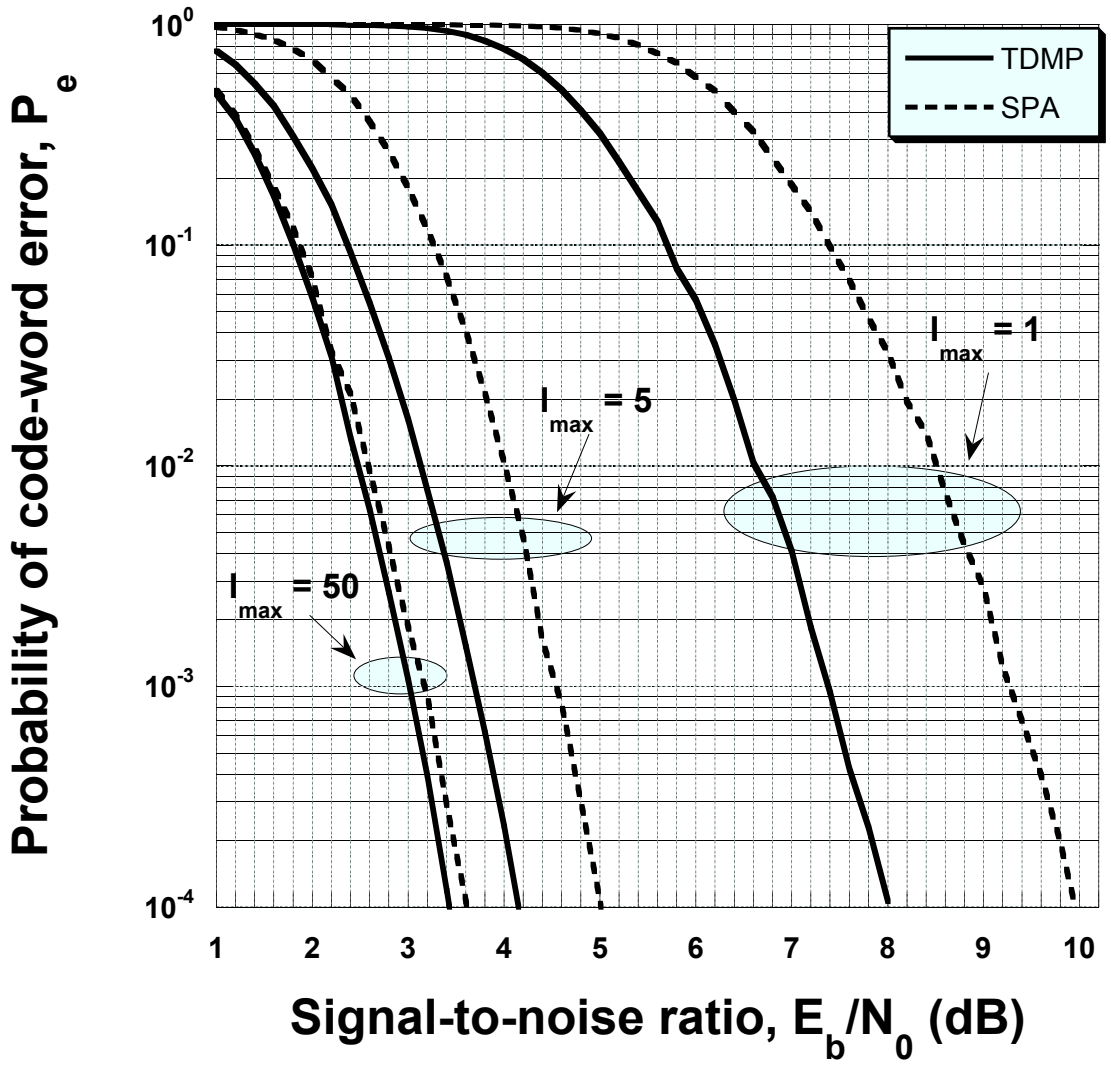

Figure 4.1: Comparison of the SPA and the TDMP algorithm for the short SFT code.

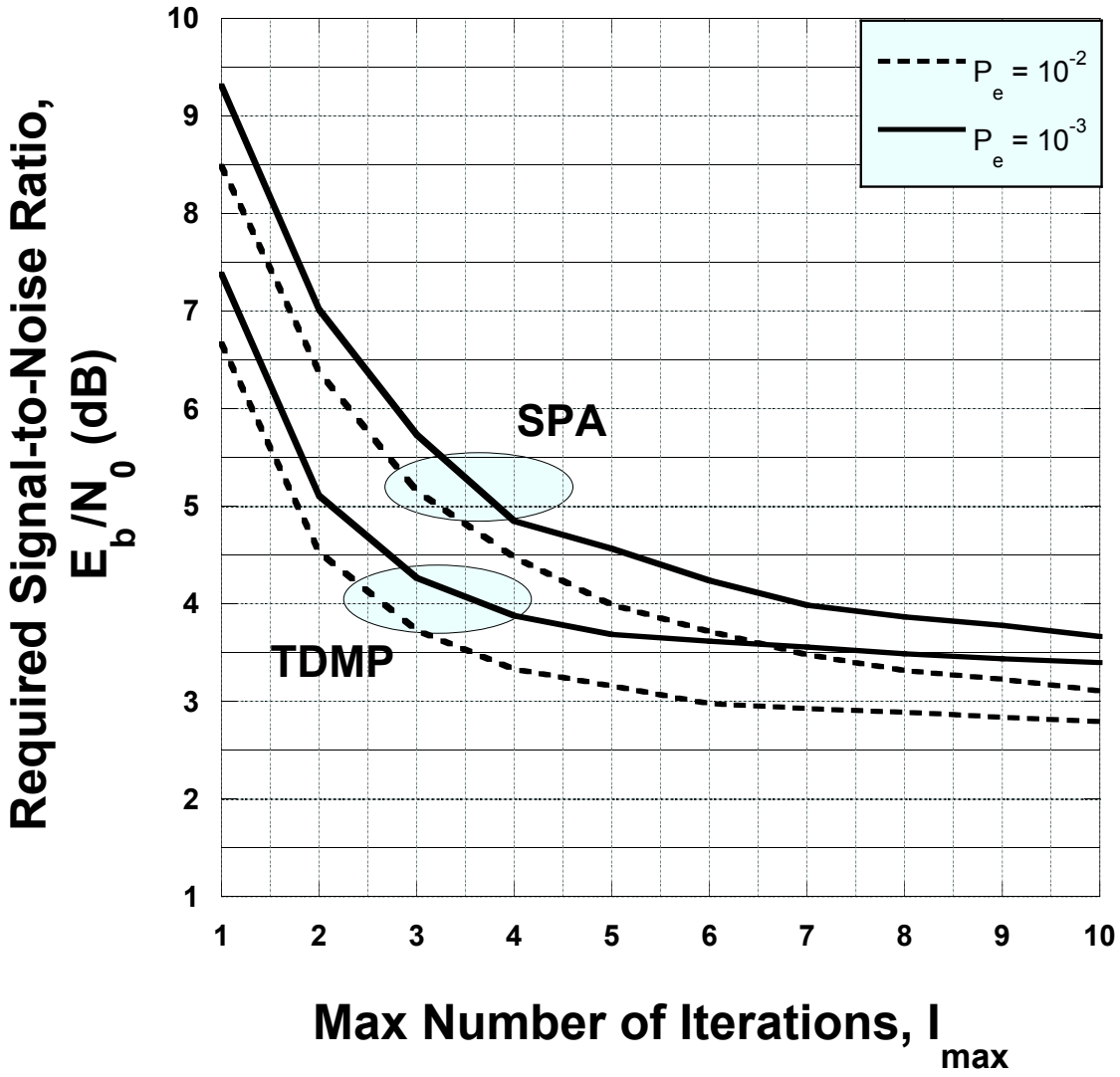

Figure 4.2: Required SNR versus I_{max} for the short SFT code.

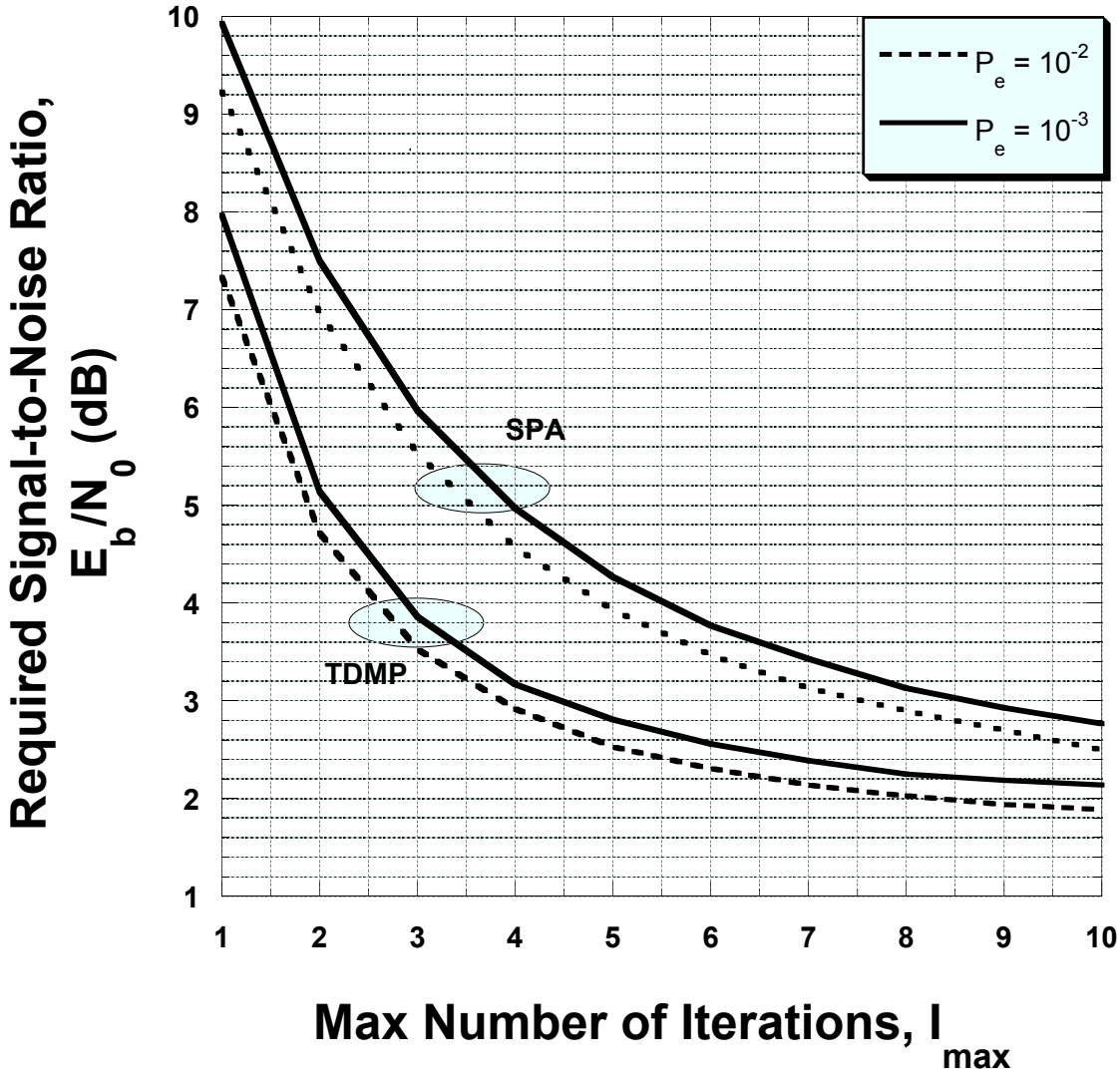

Figure 4.3: Required SNR versus I_{max} for the long SFT code.

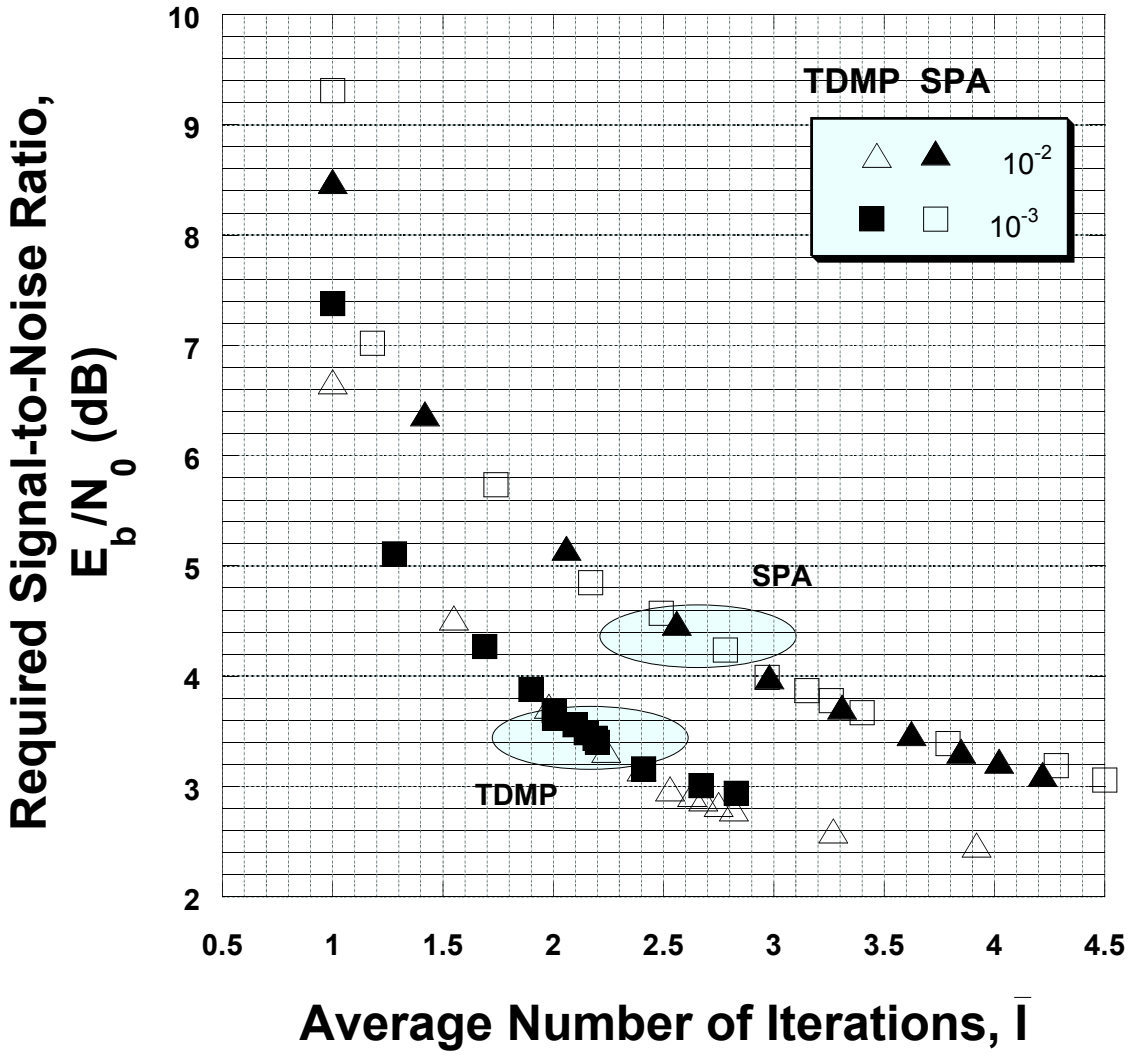

Figure 4.4: Required SNR versus $\overline{I}$ for the short SFT code.

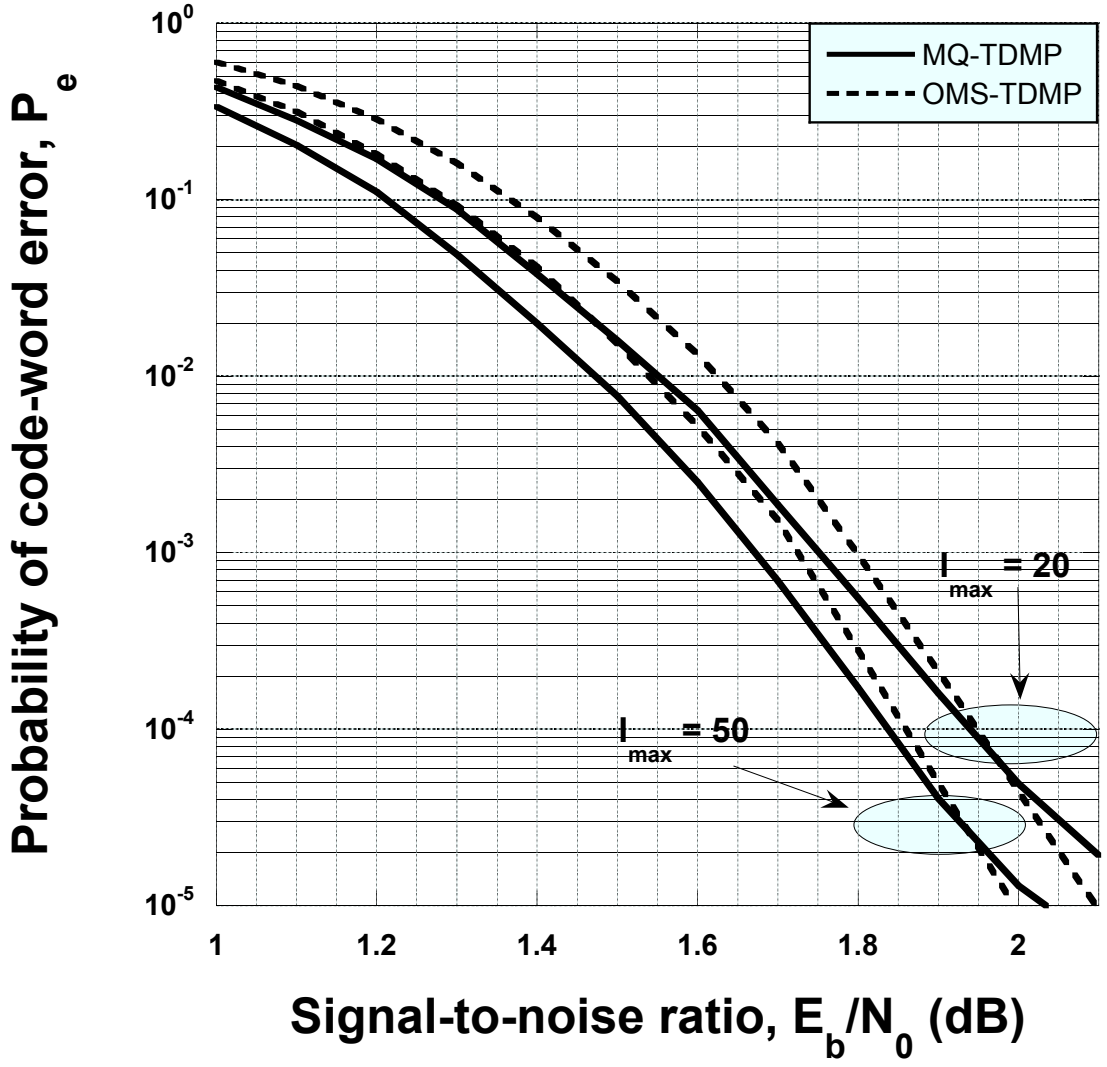

Figure 4.5: Comparison of the OMS-TDMP and MQ-TDMP algorithms for the (2304,1152) WiMAX code.

Chapter 5

SIMD Processor Architectures

Most embedded DSPs that are designed for a high degree of DLP contain a large number of functional units that perform fixed-point arithmetic. Each functional unit can perform SIMD operations on packed operands, in many instances supporting more than one software-selectable choice of its arithmetic operations. The functional units are interconnected by a high-throughput on-chip network that is designed to support high-data-rate communications between the functional units [15, 60, 61]. In this chapter we briefly describe a generic packed-data SIMD architecture supporting multiple, fixed-point data-precision modes with multiple networked functional units. We illustrate the architecture with a specific commercial processor, the Storm-1 processor [15], and we discuss the considerations that arise in implementing the TDMP algorithm on the processor.

5.1 High-performance fixed-point SIMD architectures

A SIMD architecture provides a form of parallel computing in which each functional unit performs the same operation concurrently over multiple, independent sets of operands. Operands are grouped (packed) into blocks within a register of a fixed size, and each instruction is applied separately to each operand at the same time. The SIMD architecture thus amortizes the cost of each instruction over multiple operations, potentially reducing the power consumption or chip area compared with single-data operations.

Most SIMD architectures support multiple packed-data formats, providing a software-selectable tradeoff between the level of parallelism implemented in the functional unit and the bits of precision available to represent each operand and result. This is illustrated in Fig. 5.1 for a processor supporting signed-magnitude fixed-point arithmetic. (The same concepts hold if ones complement or twos complement arithmetic is supported.) The functional unit employs registers with a width of $32N$ bits which can be used in any of three ways. If 32-bit mode is selected, each register contains N 32-bit operands (as illustrated in part a of the figure) and each SIMD instruction performs N independent operations with 32-bit results concurrently. If 16-bit mode is selected instead, each register contains $2N$ 16-bit operands (as illustrated in part b) and each SIMD instruction performs $2N$ independent operations with 16-bit results concurrently. Each SIMD instruction performs $4N$ independent operations concurrently using 8-bit operands with 8-bit results if 8-bit mode is selected (as illustrated in part c). Few DSPs support a SIMD mode of less than 8 bits since lower-precision arithmetic is suitable for a narrower range of applications and results in a lower efficiency in the circuit design of the processor.

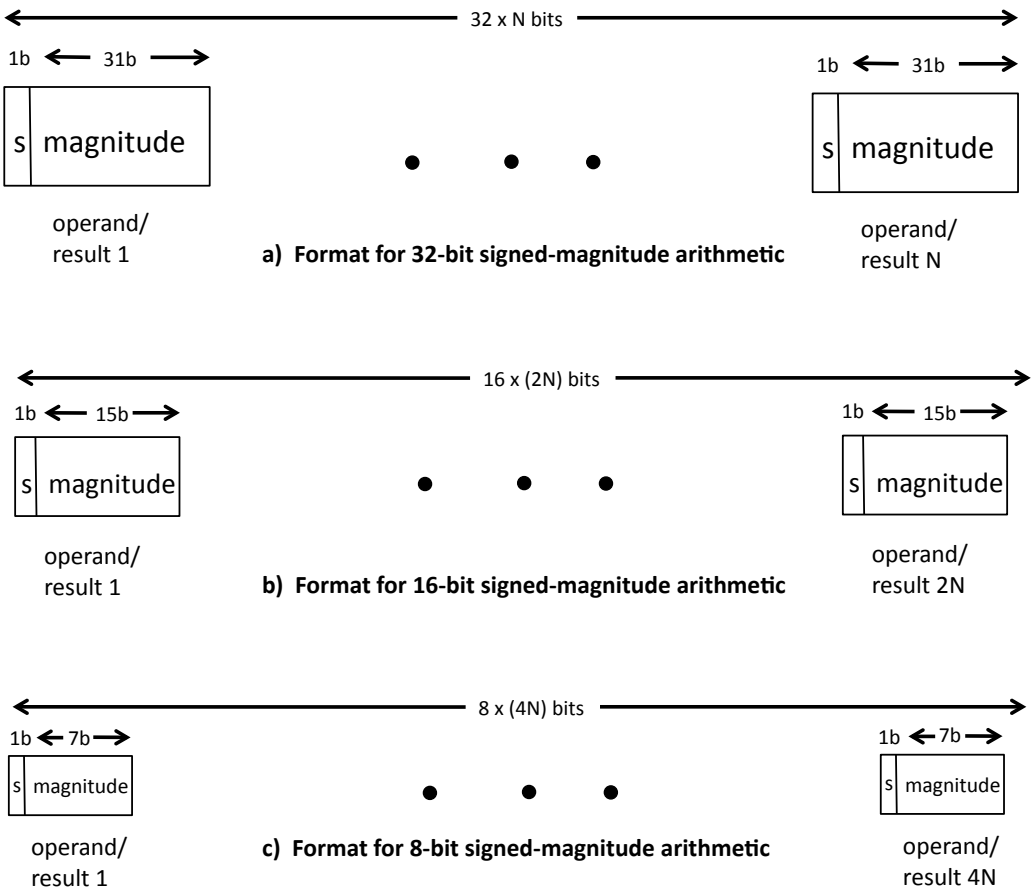

Figure 5.1: Example of packed-data formats.

The DLP can be increased beyond what is practical with a single SIMD functional unit by including multiple units within a processor. If the processor is designed so that a single instruction is issued for concurrent execution across all SIMD functional units, the aggregated functional units form a single, larger-scale SIMD unit that supports a correspondingly larger degree of DLP. Most applications require efficient communication of data between functional units, however, necessitating a means of communications among the functional units that increases in circuit complexity, power consumption, and latency as the number of functional units increases [12]. A

two-level hierarchy may be employed in which two or more SIMD functional units that are closely coupled form a processor element, and multiple processor elements are then connected via an inter-element communication network spanning the chip.

A generic processor of M network-connected processor elements containing n SIMD functional units each is illustrated in Fig. 5.2. The network connecting the processor elements can range from a relatively low-data-rate bus, which imposes a small cost in area and power consumption on the chip, to a very high-data-rate crossbar switch that is costly in area and power consumption. (In contrast, a high-date-rate interconnection among the small number of closely placed functional units within a processor element can be implemented at a low cost in many instances.)

An algorithm implemented across the processor elements that requires frequent data exchanges between the elements will suffer a communication latency that can severely impact the execution time of the algorithm if the data rate of the interconnection network is too low. For some signal-processing algorithms, the communication latency can affect performance even if a high-data-rate network is used (as discussed in Chapter 8 for TDMP decoding).

5.2 An example of a high-performance embedded SIMD processor

Modern GPUs designed for "general purpose" computing are widely known examples of SIMD-centric processors that achieve exceptionally high throughput with a wide range of signal-processing algorithms. GPUs are powerful, multi-core processors that provide a combination of task-level parallelism, thread-level parallelism, and DLP which can be used in high-throughput decoding for LDPC codes [38, 39]. GPUs

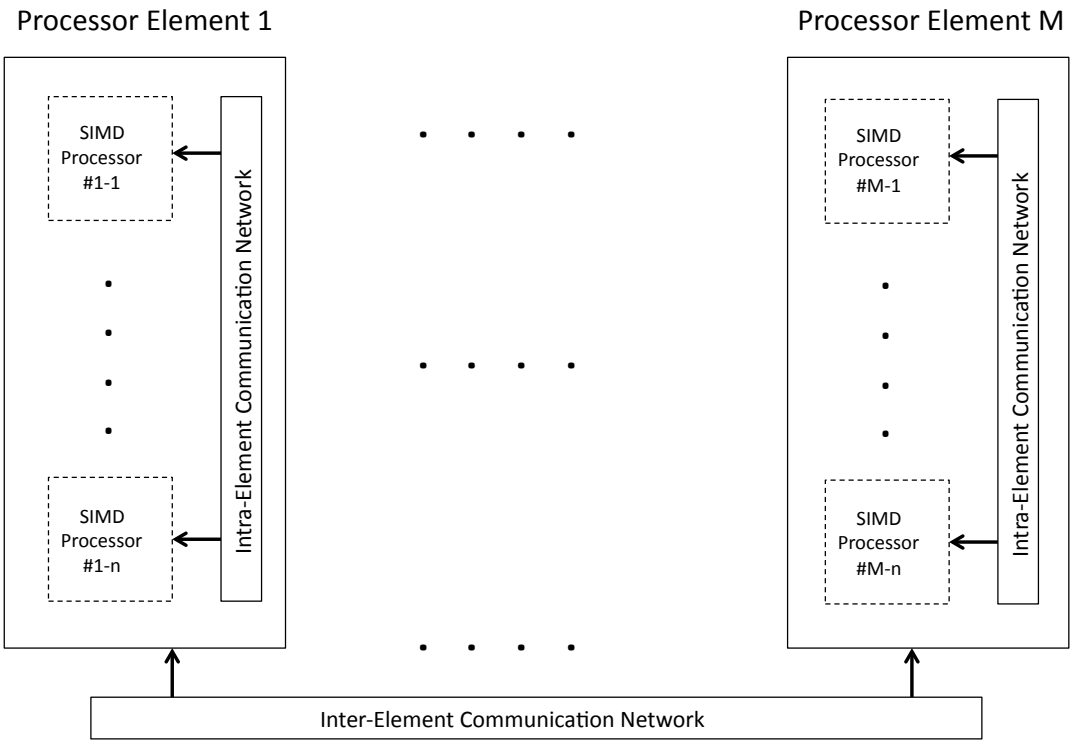

Figure 5.2: Generic SIMD architecture network.

exhibit high power consumption as a result of floating-point processing and support of highly-multithreaded task-level parallelism (including the consequent structure of the memory hierarchy [25]), however. While a GPU is a useful tool for accelerated Monte Carlo simulation of the performance of LDPC decoding, it is impractical for use in a battery-powered mobile communication device or sensor node.

An emerging alternative for computationally demanding digital signal processing is a SIMD-optimized DSP designed for embedded systems, exemplified by the stream processor. A stream processor differs from a GPU in that it is optimized for low-power embedded applications. The Storm-1 system-on-a-chip by Stream Processors, Inc., is used as the example of a stream processor in the numerical results in Chapter 8. The architecture of the chip (which is no longer in production) is

summarized below. (Details are given in [15] and a diagram is shown in Fig. 5.3.) Higher absolute throughput may be obtained with stream processors fabricated in newer process technologies using more recent architectural innovations for low-power, embedded processors with high SIMD parallelism [17, 61, 62]. The Storm-1 processor serves as a suitable platform for comparing the relative throughput obtained on a stream processor with various modifications of the TDMP decoding algorithm, however.

The Storm-1 system includes two general-purpose processors and a separate data-processing unit (DPU) with a SIMD architecture. The DPU contains 80 arithmetic logic units (ALUs) [2] organized as 16 data-parallel 5-ALU functional-unit clusters, referred to as *lanes*, that are controlled by a very long instruction word in a Harvard architecture [2]. The Storm-1 processor contains a three-level, non-caching memory hierarchy. A large off-chip memory is accessible to the DPU. Lane register files are high-bandwidth, per-lane, on-chip memory used to stage data for processing by the ALUs. Operand register files within each lane's ALU cluster, addressable only within the cluster, serve as local registers. Data can also be exchanged directly between ALU clusters using an inter-lane crossbar switch. The accompanying compiler allows an application programmer to exploit available DLP and instruction-level parallelism without the need to explicitly manage the resources of the lanes and the associated memory.

The DPU is designed for the flow of similarly formatted records of a large data set which form a *stream*. The stream is processed by one or more *kernel functions*, each of which is a compute-intensive inner loop that applies parallel processing to a stream that is resident in on-chip memory. The computations in the parallel-processing units are thus restricted to records in the stream as atomic data units and kernel functions as atomic instructions. Control tasks and computations that do not fit well within

this stream-processing paradigm are assigned by the compiler to a general-purpose coprocessor.

Analysis by the compiler establishes the stream allocation and run-time data transfers into on-chip memory for kernels and streams based on dependencies associated with execution. Parallel processing occurs within kernels only; consequently, single structured instruction flow is preserved. Kernels are managed by the compiler, and they can form pipelines which share intermediate stream results. Data reference by a kernel is limited to the data records it is processing and other locally retained constants and variables. The compile-time analysis imposes a tight control on the use of the memory hierarchy, hiding the access latency to external memory for many tasks.

The simplicity of the stream-processing application programming model is achieved at the cost of restrictions in the computation model and strict management of the memory hierarchy. The resources of the parallel-processing architecture are utilized most efficiently if the application has characteristics consistent with these constraints. Specifically, the application should exhibit compute intensity, DLP, and data locality [25]. Many signal-processing algorithms exhibit these characteristics.

The Storm-1 processor provides 8-bit packed-data and 16-bit packed-data modes of saturating fixed-point arithmetic using 32-bit data registers. The production device was fabricated in a 130nm process. It employs a nominal DPU clock rate of 800 MHz, resulting in an aggregate computation rate of 512 8-bit fixed-point GOPS (billion operations per second) or 256 16-bit fixed-point GOPS [15]. An evaluation board which operates at a lower clock rate is used (along with an emulator) to obtain the decoder performance measurements discussed in Chapter 8, but the results are given for the clock rate of the production device. The programming language used for application development for the Storm-1 processor is based on ANSI "C" with

enhancements that define the kernel functions and streams as well as several compiler directives (pragmas).

5.3 Implementation of the TDMP algorithm on a stream processor

The TDMP algorithm is well suited to implementation on the Storm-1 processor if it is used with LDPC codes in which large groups of consecutive row updates can be performed in parallel without data conflicts. For example, a QC-LDPC code with circulant permutation submatrices permits a degree of DLP equal to the row dimension of the submatrices. Each block of parallel row updates forms one subiteration of a decoder iteration. In this circumstance, the algorithm is characterized by a high level of available data parallelism and a high level of data locality. The compute intensity is only moderate, but data exchanges are limited to the high-speed inter-lane crossbar switch.

A single compiler directive for the Storm-1 processor can specify up to 64 concurrent data-parallel computations (four per lane) by using its 8-bit packed-data mode. This allows simultaneous processing of four check nodes per lane which yields a total of 64 concurrent row updates across the 16 lanes of the processor. Thus the row-update DLP available with the compiler and the architecture in 8-bit mode is fully exploited if the dimension of the permutation submatrices is an integer multiple of 64.

The five ALUs per lane also provide the compiler with the opportunity for instruction-level parallelism in the calculations associated with each row update, including parallel updates of as many as five posterior values for each row in Steps 2–6

of the algorithm. Since most of the QC-LDPC codes of interest have a parity-check matrix with a weight of five or more for most rows, an assignment of four concurrent row updates to each lane should result in a high utilization of the processor's 320 ALUs in 8-bit mode. (The diagnostic tools provided with the Storm-1 evaluation board do not allow measurement of the VLIW packing ratio or the ALU utilization to assess the average level of instruction-level parallelism achieved by the decoding algorithm, however.)

As a result of the match between the available parallelism in the LDPC code's parity-check matrix and the degree of DLP supported by the processor, straightforward programming of the TDMP decoding algorithm results in an implementation in which extrinsic messages and posterior updates for a given row of the parity-check matrix are managed by the same lane of the processor throughout the decoding of a received word. Consequently, information defining the variable-node participation in a given row is loaded once into the operand register files of the responsible lane and retained for the duration of decoding. The structure of the parity-check matrix permits its representation in a form amenable to efficient loading and storing, which we exploit (and which is exploited elsewhere in implementations on GPUs [33, 39, 40]).

The posterior value for each variable node must be communicated to another lane after it is updated in a subiteration, in general, where the recipient lane is the one that requires the value soonest in a future subiteration. Most of the latency in the data transfers can be hidden during the message-passing phase (Steps 2–7) of the TDMP algorithm due to the computation required during the phase. The same posterior transfers must occur during the parity-check phase, however, and the lower computational load in that phase exposes most of the latency as a decoding delay. As shown in Chapter 8, the exposed inter-lane communication latency can be a significant factor in limiting the throughput of the TDMP decoder using the standard schedule

consisting of separate message-passing and parity-check phases in each iteration.

Imperfect regularity in the data structures employed by the TDMP decoder can result in conditional execution and branching which markedly reduces the average utilization of the processor's resources. These problems arise if the LDPC code is irregular in the row weights of the parity-check matrix, such as the WiMAX codes considered in the examples in Chapter 8. (The same observation is noted in [39] regarding decoding irregular LDPC codes on a GPU.) We address this problem by adding "dummy" variable-node positions to the representation of the parity-check matrix and corresponding dummy circulant permutation submatrices to each row-block containing fewer than the maximum number of non-zero submatrices. The apparent row-weights of the code are consequently "regularized", which eliminates the need for conditional execution of instructions in key parts of the code.

Each dummy variable node is initialized with a prior value set to the positive value of greatest possible magnitude (127) in its 8-bit representation, corresponding to a polarity of zero with high reliability. No two row-blocks contain non-zero dummy circulant submatrices in the same column positions, so each dummy variable node participates in message passing for only one row. The technique reduces the computation time per iteration at the cost of a modest increase in the size of the representation of the parity-check matrix. Simulations show it has no measurable detrimental effect on the probability of code-word error or the convergence time of the decoding algorithm.

The TDMP decoder is implemented as a single kernel function on the Storm-1 processor. Each record in the input stream corresponds to the vector of quantized channel outputs for one received word, and each record in the output stream corresponds to one detected code word or an indication of decoder failure. The instruction kernel and the information defining the code's parity-check matrix are loaded from

the global memory into the DPU only once, at the start of decoding; both are retained in the DPU throughout the processing of many consecutive streams (i.e., while decoding many received words). In our implementation, the transfer of the channel samples to the DPU's lane register files from the general-purpose processor occurs while the DPU is otherwise idle. The resulting latency is a negligible fraction of the decoding time for a received word.

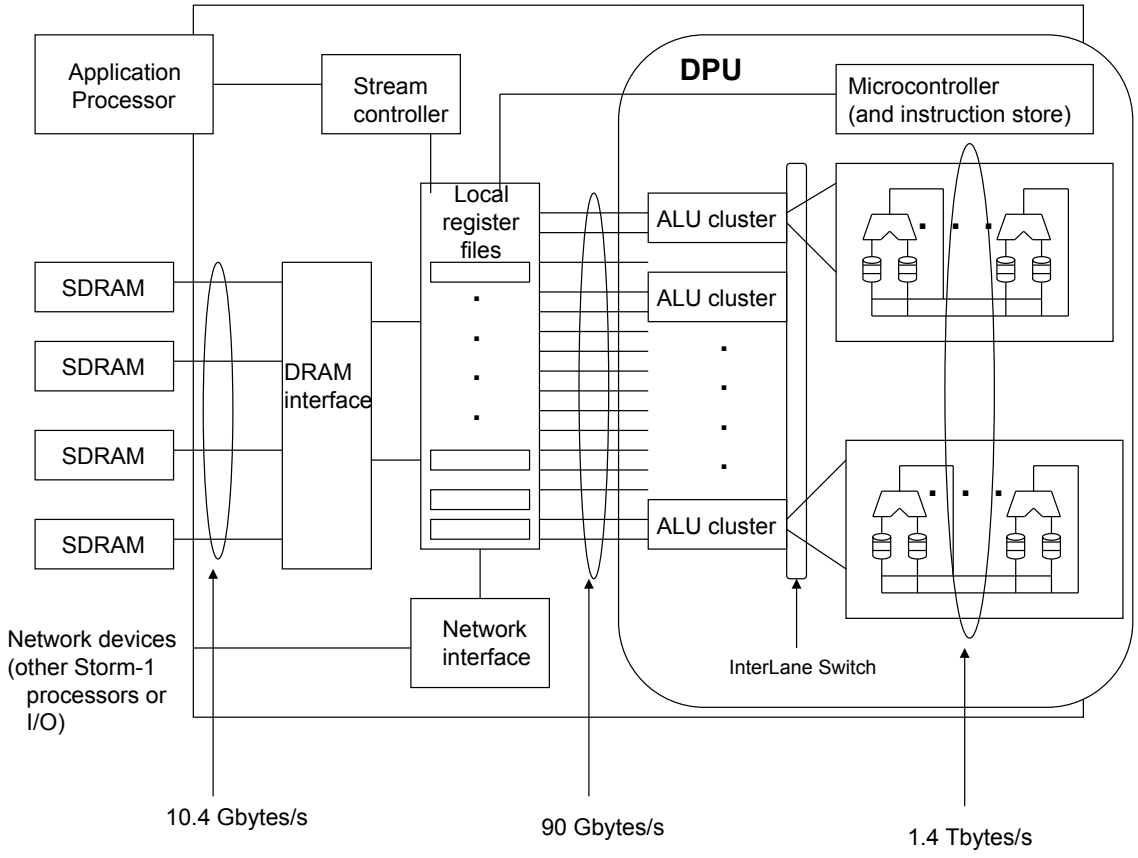

Figure 5.3: Storm-1 processor architecture (adapted from [15]).

Chapter 6

OMS-TDMP Decoding with Low-Resolution Arithmetic

In this chapter, we examine the effect of the resolution on the performance of the OMS-TDMP algorithm using fixed-point saturating arithmetic. The sensitivity of the algorithm to the resolution is compared with the sensitivity of the OMS-SPA, and the effect of the code's rate and block length on the sensitivity is considered. A simple technique to reduce the effect of saturation on OMS-TDMP decoding is described, and its effectiveness is investigated.

6.1 Effect of the precision on fixed-point OMS-TDMP decoding

The effect of the precision on the performance of the fixed-point OMS-TDMP algorithm is illustrated by a comparison with its effect on the performance of the OMS-SPA for the (2304,1152) WiMAX LDPC code. The probability of code-word error is shown as a function of the signal-to-noise ratio for both algorithms in Fig. 6.1

with both 8-bit precision and 16-bit precision. The performance is shown for 10, 20, and 50 iterations of each algorithm. In each instance, the resolution and offset is chosen to minimize the probability of code-word error at a signal-to-noise ratio of 2 dB using 20 iterations of the algorithm.

Both 16-bit OMS-TDMP decoding and 16-bit OMS-SPA decoding result in performance that is within 0.1 dB of the respective floating-point algorithms (not shown in the graph). The performance of the OMS-TDMP algorithm is much better than the performance of the OMS-SPA for a given number of iterations with 16-bit decoding. The OMS-TDMP algorithm achieves performance with 10 iterations that is nearly equal to the performance of the OMS-SPA with 20 iterations, and OMS-TDMP decoding requires only 20 iterations to achieve performance similar to that of OMS-SPA decoding with 50 iterations. Note that the performance of the OMS-TDMP algorithm improves substantially (by 0.32 dB at a probability of code-word error of 10^{-4}) if the maximum number of iterations is increased from 10 to 20; thus, there is strong motivation for employing the OMS-TDMP algorithm with the larger number of iterations.

Reducing the precision from 16 bits to eight bits has quite different consequences for the OMS-TDMP algorithm and the OMS-SPA. As seen in Fig. 6.1, the performance of 8-bit OMS-SPA decoding is indistinguishable from the performance of 16-bit OMS-SPA decoding for a given number of iterations. In contrast, the reduction in precision results in a pronounced error floor with the OMS-TDMP algorithm. Consequently, the performance at a probability of code-word error of 10^{-4} is degraded by 0.02 dB, 0.16 dB, and 0.23 dB for 10 iterations, 20 iterations, and 50 iterations, respectively, when the precision is reduced to eight bits. For a probability of code-word error of 10^{-5}, the corresponding degradation is 0.28 dB, 0.72 dB, and 0.82 dB, respectively.For a sufficiently high signal-to-noise ratio, the performance of 8-bit

OMS-TDMP decoding is poorer than the performance of 8-bit OMS-SPA decoding.

The relationship between the arithmetic precision and the choice of the OMS-TDMP algorithm's parameters is illustrated in Fig. 6.2 in which the performance is shown for 8-bit and 16-bit OMS-TDMP decoding and various choices of the quantizer resolution and the offset used in the decoding algorithm. Results are shown for the (2304,1152) WiMAX LDPC code and a maximum of 20 decoding iterations per received word.

The performance of 16-bit decoding with an offset of 0.125 is essentially the same with a resolution of either 0.0625 or 0.125. Either combination of parameters for 16-bit decoding result in performance within 0.1 dB of the performance of the floating-point decoder (which is not shown) that uses the offset selected to minimize the probability of code-word error at a signal-to-noise ratio of 2 dB. An offset of 0.125 is close to the optimal value for 16-bit decoding, and 16-bit precision permits a combination of the resolution and the range for which quantization noise and saturation both have a negligible impact on the decoder's performance. In contrast, a resolution and an offset of 0.25 results in a 0.22 dB degradation in the performance of 16-bit decoding. Even though the offset of 0.25 is optimal for use with a resolution of 0.25, it is highly sub-optimal for a smaller resolution. (Recall that the offset must be an integer multiple of the resolution for fixed-point decoding.)

The use of 8-bit decoding results in a severe tradeoff between the frequency of arithmetic saturation and constraints on the choice of the offset. If the resolution is small, the range is too restricted and the performance is severely affected by saturation. This is seen in the performance of 8-bit decoding with a resolution of 0.0625 and an offset of 0.125; it results in a high error floor. If the resolution is large, in contrast, the decoder is constrained to using a suboptimal offset (as noted above for 16-bit decoding), resulting again in a loss of 0.25 dB across the full range of signal-to-

noise ratios compared with optimal 16-bit decoding. If the 8-bit decoder employs the intermediate resolution of 0.125 and an offset of 0.125, its performance is within 0.05 dB of the performance of optimal 16-bit decoding for signal-to-noise ratios below 1.8 dB. At higher signal-to-noise ratios, however, the occurrence of saturation results in an error floor that results in much poorer performance than optimal 16-bit decoding.

The effect of the precision on the performance of OMS-TDMP decoding is more pronounced with a long code than with a short code. This is illustrated in Fig. 6.3 for rate-1/2 codes of three block lengths: the WiMAX codes of block lengths 2304 and 1536 and a "WiMAX like" code of block length 1008. (The latter code uses the same base matrix and circulant-shift equation as the rate-1/2 WiMAX codes but with circulant permutation submatrices of dimension 42.) The decoder uses a maximum of 20 iterations per received word with each code.

The performance of 8-bit decoding at a probability of code-word error of 10^{-5} is 0.68 dB poorer than the performance of 16-bit decoding with the block-length-2304 code, it is 0.42 dB poorer with the block-length-1536 code, and it is only 0.12 dB poorer with the block-length-1008 code. Moreover, the performance with the shortest code is better than the performance with either of the two longer codes if the signal-to-noise ratio is above 2.8 dB. The base matrix, the row-weight distribution, and column-weight distribution are identical for the parity-check matrices of all three codes. The longest code has a larger number of channel symbols than the other two codes and requires a larger average number of iterations to achieve a given error probability, however; thus its decoding includes the execution of a larger total number of row updates with correspondingly more opportunities for saturation of posterior values.

The effect of the precision on the performance of OMS-TDMP decoding is also more pronounced with a code of a lower rate than with a code of a higher rate. This is

illustrated in Fig. 6.4 for three block-length-2304 WiMAX codes: the rate-1/2 code, the rate-2/3 "A" code, and the rate-3/4 "A" code. The use of 8-bit precision results in an error floor that is discernible for probabilities of code-word error below 10^{-3} with the rate-1/2 code, for probabilities of code-word error below 1.5×10^{-4} for the rate-2/3 code, and for probabilities of code-word error below 1.6 x 10^{-5} for the rate-3/4 code. The resulting performance difference between 16-bit decoding and 8-bit decoding is 0.68 dB, 0.17 dB, and 0.02 dB at a probability of code-word error of 10^{-5} for the codes of rate 1/2, 2/3, and 3/4, respectively. The sensitivity to the decoder's precision is greater for codes with greater error-correcting capability, in general, as illustrated in both Fig. 6.3 and Fig. 6.4.

6.2 Constrained extrinsic updates

The examples in the previous section illustrate the need to use a small enough resolution Δ to permit the use of the optimal offset η in order to achieve acceptable performance at a low-to-moderate signal-to-noise ratio with the OMS-TDMP algorithm. The limited range that results leads to frequent arithmetic saturation, however, if 8-bit precision is used.

A posterior value is desensitized to reinforcing updates while it is saturated or close to saturation. Thus the outcome of a sequence of updates depends on the order in which the updates occurs as well as their actual values. For example, the ordered sums 127-127+127-127+64 and 127+127-127-127+64 are both equal to +64 in high-precision saturating arithmetic. In signed, 8-bit saturating arithmetic, however, the first ordered sum results in a value of +64 whereas the second ordered sum results in a value of -63.

The effect of order dependence on the outcomes for the posterior values can

be reduced by limiting the size of any single update to less than the maximum size allowed by the precision. We implement this by constraining the magnitude of each extrinsic message calculated in the step 4 of the TDMP algorithm in Chapter 4 so that steps 5 and 6 in the algorithm are replaced with the following:

5) Limit the maximum extrinsic updates in step 4 to

$$\tilde{\lambda}_j^i = \text{sgn}[\lambda_j^i] \cdot \min\left(|\lambda_j^i|, \epsilon\right)$$

where ϵ is a predetermined constant that is used to limit saturation in the posteriors. It is also an integer multiple of the quantization interval.

6) Update the posterior values for the bit positions of I_i as

$\mathbf{w}(I_i) = \boldsymbol{\rho} + \underline{\tilde{\boldsymbol{\lambda}}}^{\boldsymbol{i}}$.

7) Steps 2-6 represent a decoding subiteration for one row of $\mathbf{H}$. Repeat these steps for each row.

6.3 Fixed-point OMS-TDMP decoding with an update constraint

The effect of a constraint on the magnitude of each extrinsic value is illustrated in Fig. 6.5 for the rate-1/2 WiMAX code of block length 2304. The performance is shown for both 8-bit OMS-TDMP decoding and 16-bit OMS-TDMP decoding with no constraint and for 8-bit decoding with four values of the constraint ϵ: 1.25, 2.5, 3.75, and 5.0. (The optimal constraint for 16-bit decoding results in performance

indiscernible from the performance of unconstrained 16-bit decoding.) A resolution of 0.125 and an offset of 0.125 are used in each instance.

As discussed in Section 6.1, the performance of 8-bit decoding with no constraint deviates considerably from the performance of 16-bit decoding as the signal-to-noise ratio increases above 1.8 dB. If a constraint of $\epsilon = 1.25$ is employed, then each extrinsic value is limited to the ten smallest of the 128 magnitudes that can be represented in a signed, 8-bit integer. The constraint excessively limits the decoder's ability to reflect confidence in the polarity of a code symbol, resulting in very poor performance. If the constraint is increased to $\epsilon = 0.25$, the performance of 8-bit decoding is similar to its performance without a constraint. If, instead, a constraint of $\epsilon = 3.75$ is employed, the algorithm limits the occurrence of saturation effectively without preventing the decoder from communicating a high level of confidence through a single extrinsic message. Consequently, the performance of 8-bit OMS-TDMP decoding with $\epsilon = 3.75$ is within 0.01 dB of the performance of 16-bit decoding for all probabilities of code-word error above 10^{-5}. Larger values of ϵ fail to limit saturation effectively with 8-bit decoding; thus, the performance degrades as ϵ is increased above 3.75. The average number of iterations for the 16-bit decoding and 8-bit decoding with an optimal constraint differ by a small fraction of 1% for a given value of I_{max} and a given signal-to-noise ratio.

The performance of fixed-point OMS-TDMP decoding is shown in Fig. 6.6 for a more powerful LDPC code: the DVB-S2 inner code of rate 4/9 and block length 16,200 (the "short frame" DVB-S2 LDPC code designated with the "rate 1/2" identifier in the standard [8]). The resolution and offset are both 0.125 in each instance. The results are obtained from sequential execution of the row updates so that any loss in performance arising from partial parallelization with the DVB-S2 codes [56] is not reflected.

The performance of 8-bit OMS-TDMP decoding with no constraint exhibits an error floor that is discernible for any probability of code-word error less than 10^{-2}. It results in performance that is 0.79 dB poorer than the performance of 16-bit OMS-TDMP decoding at a probability of code-word error of 10^{-3}. The loss in performance is more than twenty times the degradation observed at the same error probability with the rate-1/2 WiMAX code of block length 2304, illustrating once again that the sensitivity to the precision increases with the error-correction capability of the code.

The selection of a constraint on the magnitude of extrinsic values can be used for this code as well to optimize the tradeoff between the message-passing capability of the algorithm and its immunity to the effects of arithmetic saturation. The performance of 8-bit constrained-update OMS-TDMP decoding is shown in Fig. 6.6 for four choices of the constraint. It is seen that a constraint of $\epsilon = 6.25$ results in performance that within 0.01 dB of the performance of 16-bit decoding for any probability of code-word error greater than 10^{-4}.

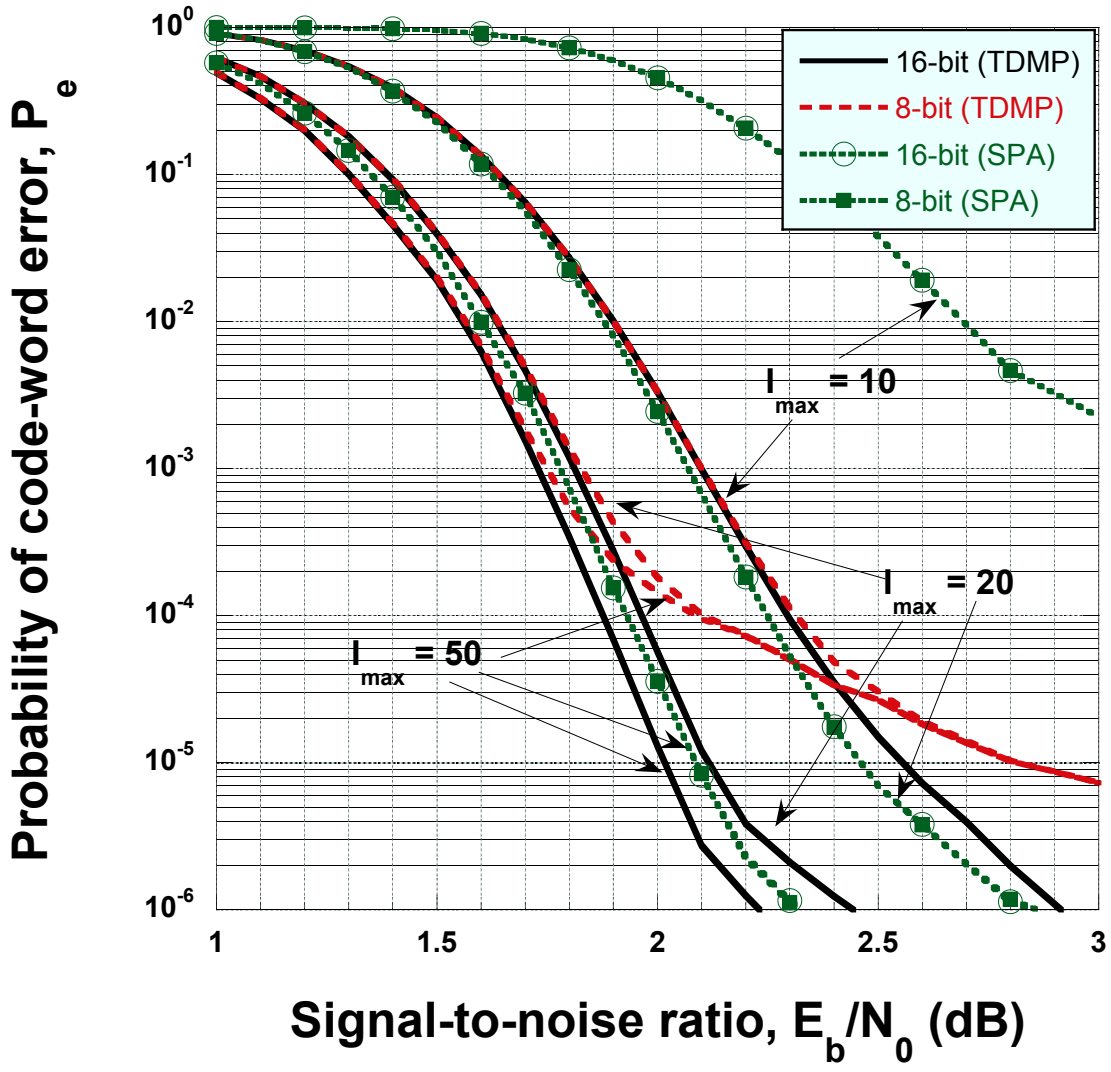

Figure 6.1: Comparison of fixed-point OMS-SPA and OMS-TDMP decoding.

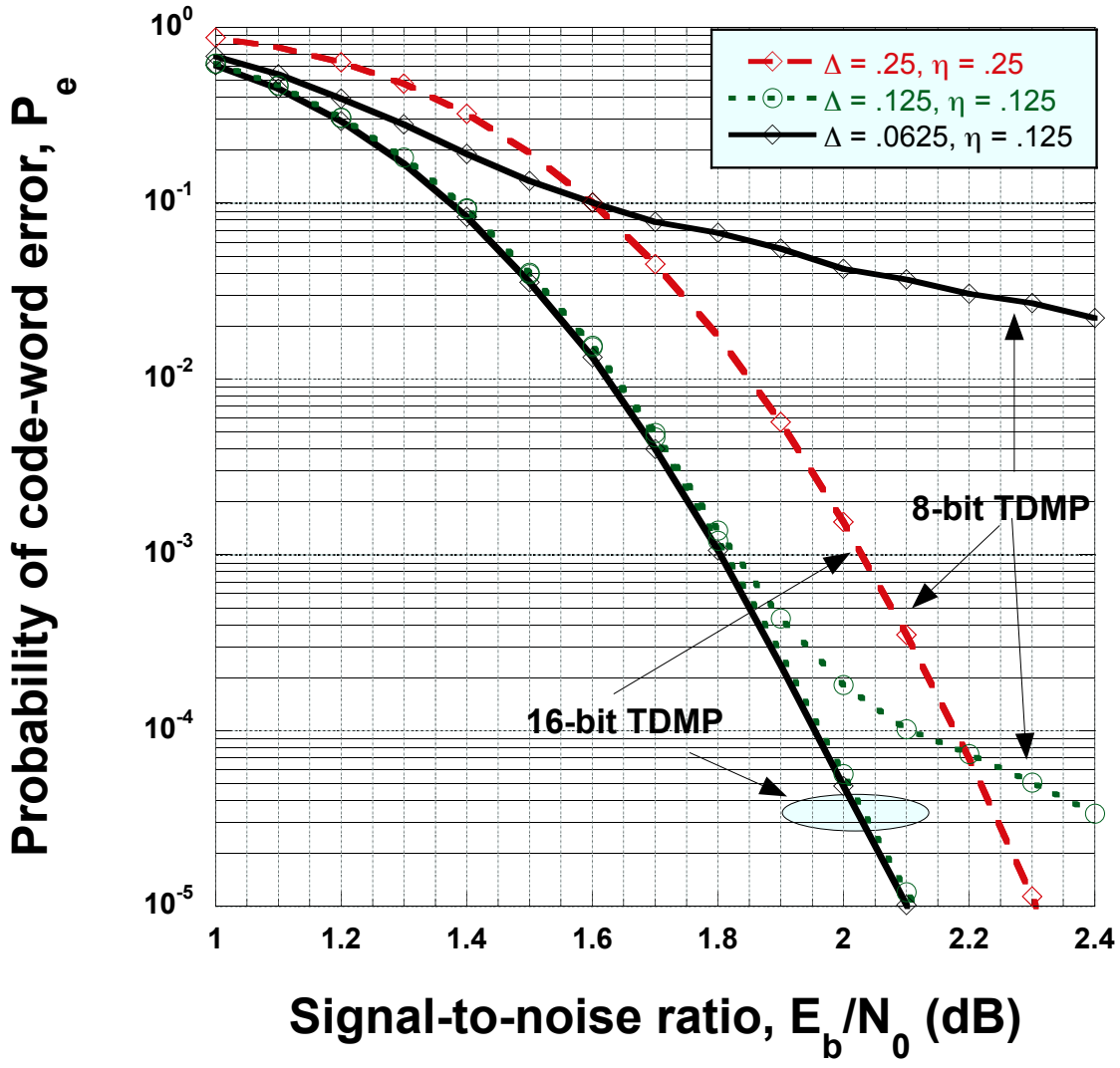

Figure 6.2: Performance of OMS-TDMP decoding with various decoder parameters.

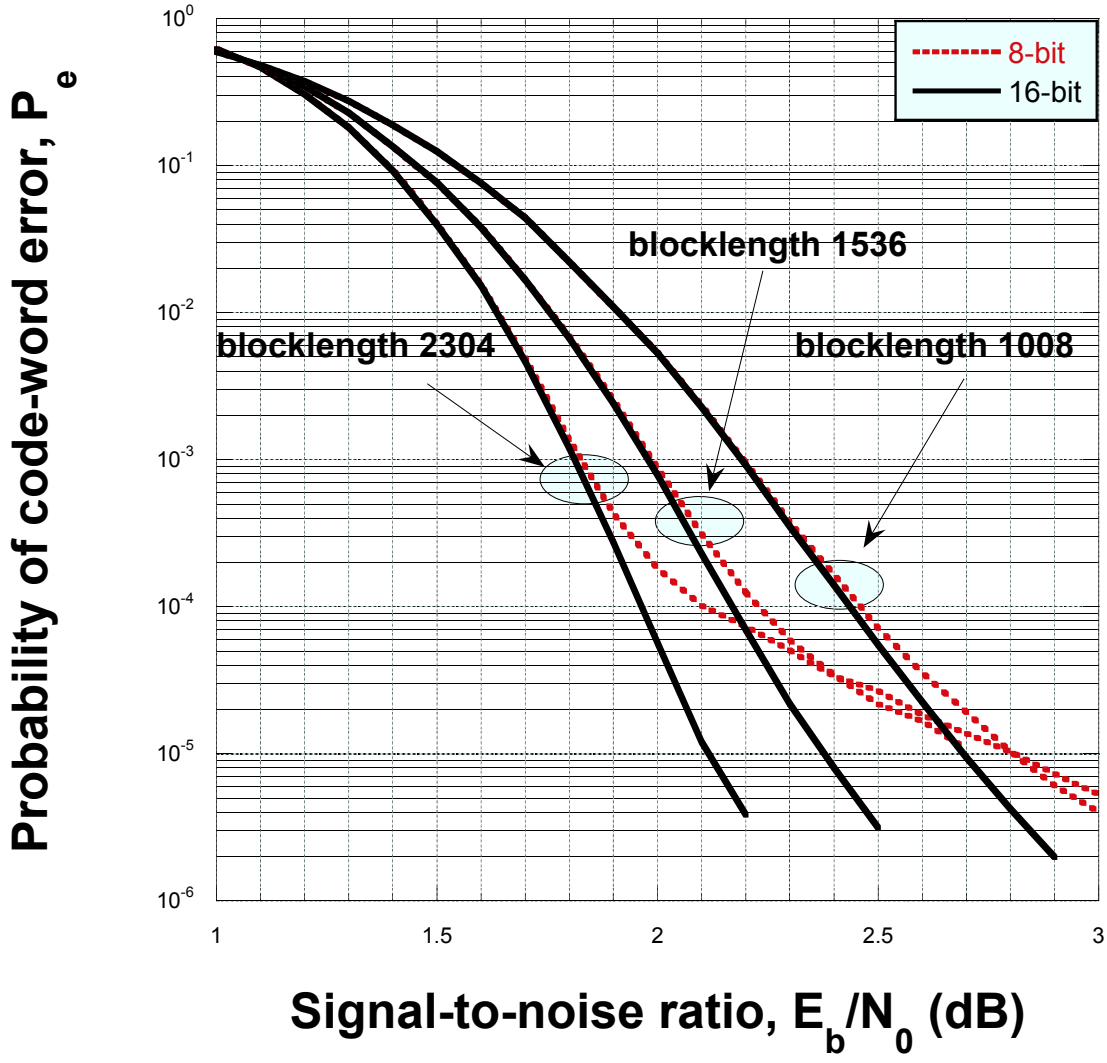

Figure 6.3: Performance of OMS-TDMP decoding for various block lengths.

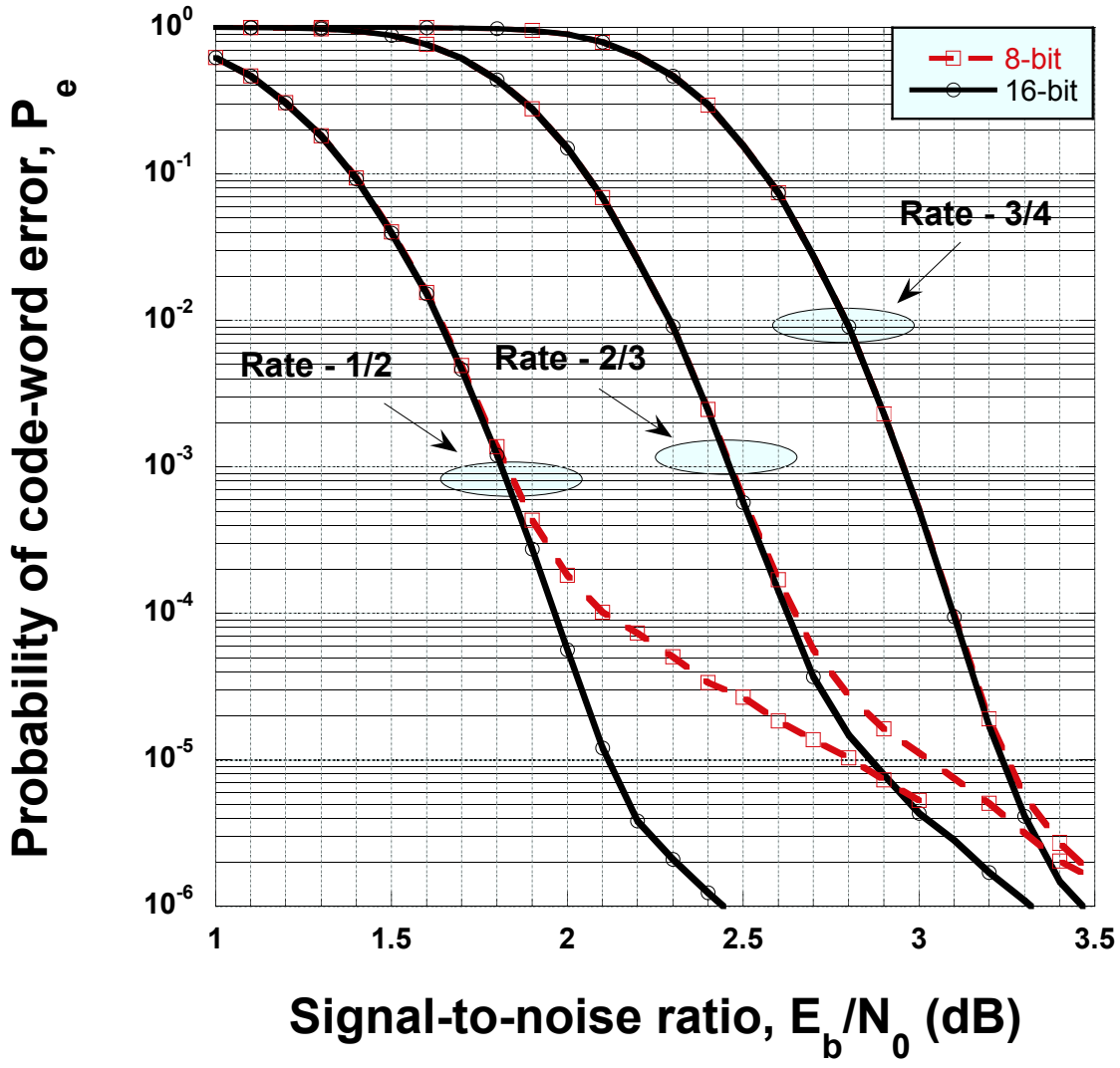

Figure 6.4: Performance of OMS-TDMP decoding for various code rates.

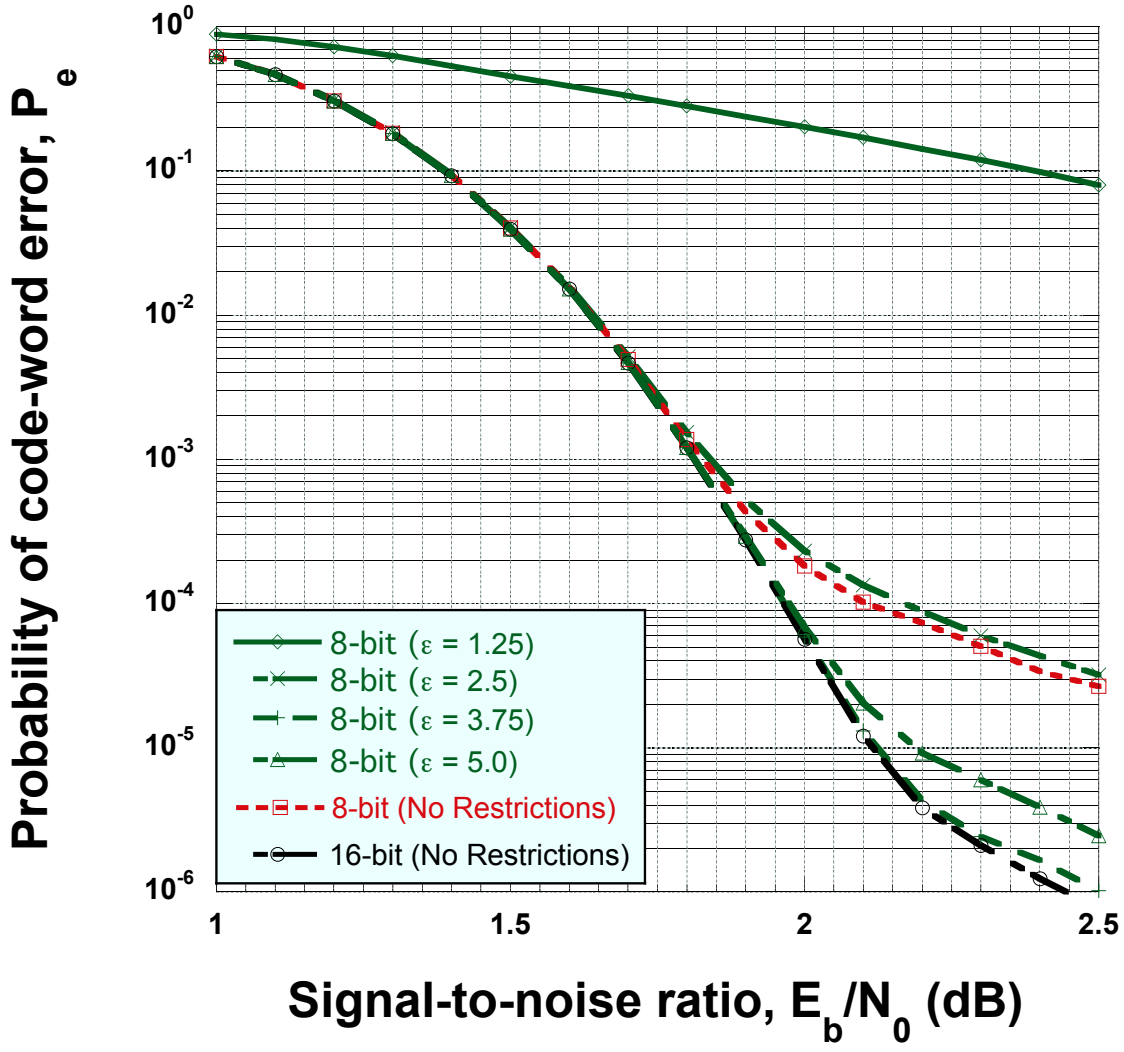

Figure 6.5: Performance with an update constraint for the (2304,1152) WiMAX code.

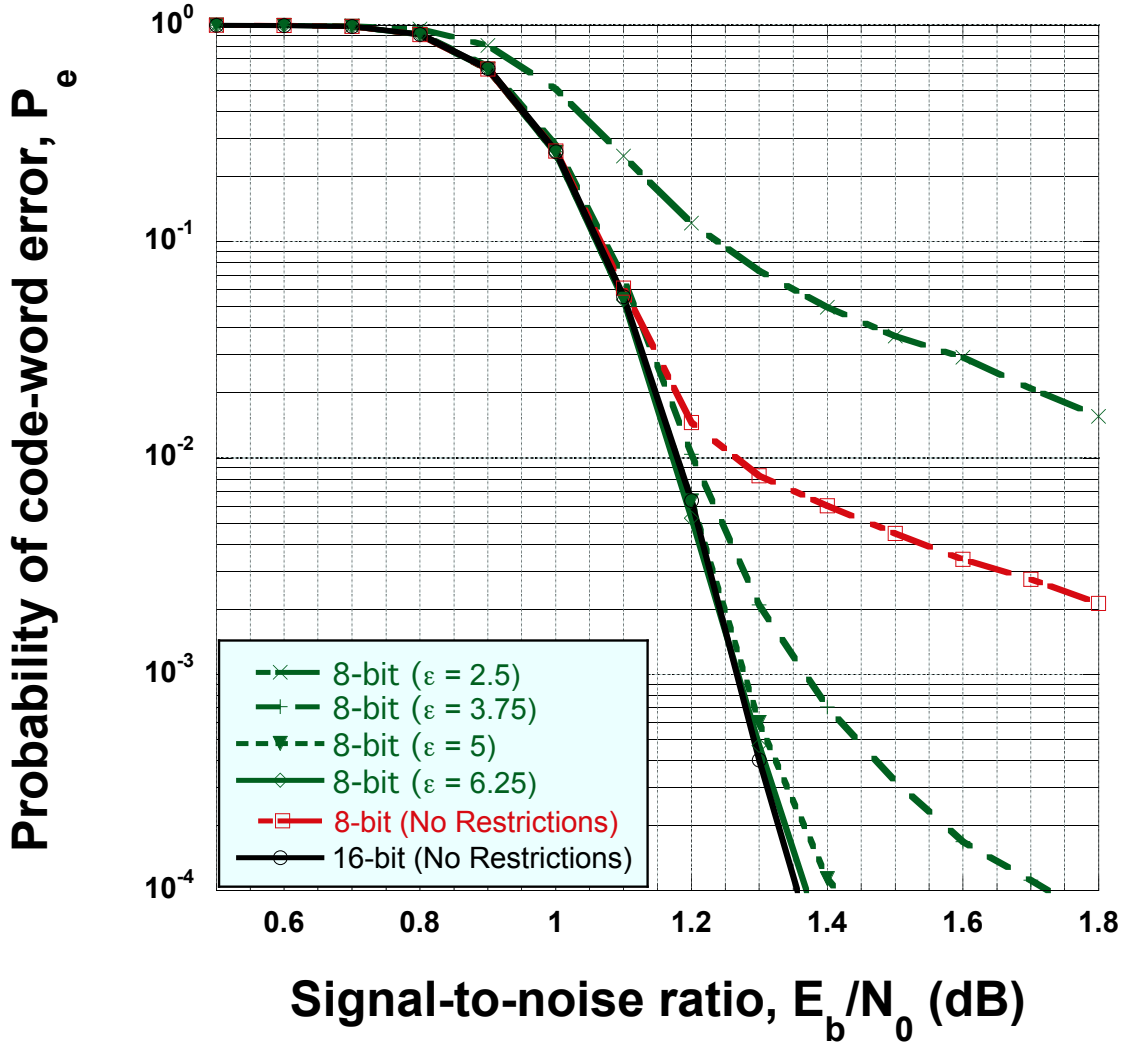

Figure 6.6: Performance with an update constraint for the (16200,7200) DVB-S2 code.

Chapter 7

MQ-TDMP Decoding with Low-Resolution Arithmetic

In this chapter, we examine the effect of the resolution on the performance of the MQ-TDMP algorithm using fixed-point saturating arithmetic. The sensitivity of the algorithm to the resolution is compared with the sensitivity of the MQ-SPA, and the effect of the code's rate and block length on the sensitivity is considered. The effectiveness of the constrained update introduced in Chapter 6 is considered for the MQ-TDMP algorithm. A modification that improves its effectiveness with the MQ-TDMP algorithm is investigated. Perfect estimation of the signal-to-noise ratio is assumed in the results shown here. (The effectiveness of the modification in the presence of estimation error is examined in [63].)

7.1 Effect of the precision on fixed-point MQ-TDMP decoding

The effect of the precision on the MQ-SPA and the MQ-TDMP algorithm is compared in Fig. 7.1 for three choices of the maximum number of iterations: 10, 20, and 50. For both algorithms, the resolution is 0.25 and $C_Q = 0.5$. The performance of 16-bit decoding exhibits negligible effects of saturation for both algorithms over the range error probabilities shown in the graph. The MQ-SPA requires about twice as many iterations to achieve a given performance as the MQ-TDMP algorithm with 16-bit decoding.

The MQ-SPA exhibits no discernible loss in performance if the resolution is reduced from 16 bits to 8 bits, regardless of the number of iterations. In contrast, the reduction in resolution has a significant effect on the performance of the MQ-TDMP algorithm. The performance with 8-bit decoding diverges from the performance with 16-bit decoding for error probabilities below 10^{-1} regardless of the maximum number of iterations. The error floor with 8-bit MQ-TDMP decoding is approximately the same regardless of the maximum number of iterations, well above 10^{-2}, but the degradation relative to 16-bit decoding is greater for the larger values of I_{max}.

The effect of the resolution on the performance of the MQ-TDMP algorithm is shown in Fig. 7.2 for both 8-bit and 16-bit precision and three values of the resolution: 0.0625, 0.125, and 0.25. For both 8-bit decoding and 16-bit decoding, the optimal value of C_Q is determined through simulation to be 0.5 if the resolution is 0.25 and 0.625 if the resolution is either 0.0625 or 0.125. (Recall that a value of $C_Q = 0.625$ can not be used with a resolution of 0.25 for fixed-point arithmetic since it is not an integer multiple of 0.25.)

The performance of 16-bit decoding is approximately the same with a reso-

lution of either 0.0625 or 0.125, with a loss in performance of less than 0.01 dB if the smaller resolution is used. Both yield approximately the same performance as floating-point decoding with its optimal value of C_Q. Thus either choice of resolution provides the combination of accuracy and range with 16-bit decoding to closely approximate the effect of arithmetic using unclipped, continuous-valued operands. An increase in the resolution to 0.25 results in a loss in performance of about 0.05 dB, however, and further increases in the resolution cause a further decline in performance due to the loss of accuracy resulting from the larger resolution.

The performance of 8-bit MQ-TDMP decoding is much poorer than 16-bit decoding regardless of the resolution. Each choice of resolution produces either a significant loss of accuracy or a significant loss of range (with resulting saturation) in the approximation of real-values operands; consequently a high error floor results in each instance. The best performance of 8-bit decoding occurs if a resolution of 0.25 is used and it degrades as the resolution is decreased (due to the increased effect of saturation). Thus the relationship between the resolution and performance for 8-bit decoding is counter to the relationship for 16-bit decoding, and the sensitivity is much greater for 8-bit decoding than for 16-bit decoding. More importantly, the performance of 8-bit decoding is much poorer than the performance of 16-bit decoding if each is used with its optimal parameters.

The performance of 8-bit and 16-bit MQ-TDMP decoding is shown in Fig. 7.3 for the same three rate-1/2 WiMAX and "WiMAX-like" codes considered in Chapter 6. A resolution of 0.25 and a value of 0.5 for C_Q is use for both decoders. Once again, the performance of 8-bit decoding is markedly poorer than the performance of 16-bit decoding due to the frequent occurrence of arithmetic saturation. The loss in performance is less pronounced for codes of shorter block length. For a probability of code-word error of 10^{-2}, for example, the performance for the code of block length

2304 is 1.0 dB poorer with 8-bit decoding than with 16-bit decoding. The difference in performance is only about 0.3 dB for the code of block length 1008 and the same error probability. All three codes have parity-check matrices with the same row and column weight distributions, so the differences in performance are attributable to the different block lengths.

The effect of the resolution on the performance of the MQ-TDMP algorithm also depends on the code rate. The dependence is illustrated in Fig. 7.4 for WiMAX codes of block length 2304 and three rates: 1/2, 2/3, and 3/4. (They are the same three rate-1/2 WiMAX codes considered in Chapter 6.) A resolution of 0.25 and a value of 0.5 for C_Q is used in each instance. For each code, the use of 8-bit decoding results in a significant loss in performance compared with 16-bit decoding, but the performance degradation is less pronounced with an increase in the code rate. The results in both Fig. 7.3 and Fig. 7.4 illustrate the general principle that the sensitivity of the performance to the resolution is greater for codes with greater error-correction capability.

7.2 MQ-TDMP decoding with a fixed update constraint

In Chapter 6, it is shown that a properly chosen fixed constraint on the magnitude of extrinsic messages largely mitigates the effect of saturation in 8-bit OMS-TDMP decoding. The effectiveness of the fixed-constraint technique with MQ-TDMP decoding is illustrated in Fig. 7.5 for the (2304,1152) WiMAX code. The resolution is 0.25 and $C_Q = 0.5$. A constraint of $\epsilon = 7.5$ results in a great improvement for 8-bit decoding compared to the absence of the constraint for a signal-to-noise ratio of

less than 2.2 dB, whereas a constraint of 10.0 yields better performance at a higher signal-to-noise ratio.

Neither choice of the fixed constraint achieves performance comparable to 16-bit decoding, however. Either constraint results in performance that is more than 0.3 dB poorer than the performance of 16-bit decoding at a probability of code-word error of 10^{-4}. The shortcoming of the fixed-constraint technique with 8-bit decoding is also apparent for the (16200,7200) DVB-S2 code. The optimal fixed constraint ($\epsilon = 10.0$) results in 8-bit performance that is poorer than the 16-bit performance by 0.05 dB at a probability of code-word error of 10^{-3} and by 0.2 dB at a probability of code-word error of 10^{-2}.

7.3 MQ-TDMP decoding with a variable update constraint

The codes considered in the examples of the previous section are column irregular (and in fact, most of the QC-LDPC codes of practical interest are column irregular). The posterior values for code symbols with a larger corresponding column weight in the parity-check matrix are subjected to more frequent updates than the posterior value associated with a smaller column weight. Consequently, the former values are more susceptible to order dependence of the updates.

The differing sensitivity of different code-symbol decisions to saturation is addressed by modifying the constraint on magnitude of extrinsic messages so that the constraint depends on the variable node to for which the message is intended. This is achieved by replacing step 5 in the constrained-update algorithm of Chapter 6 by the following:

5. For $j \in I_i$, replace λ_j^i by $\text{sgn}[\lambda_j^i] \cdot \min\left(|\lambda_j^i|, \epsilon_j\right)$, where ϵ_j is a predetermined constant that is used to limit fixed-point saturation in the posterior value for v_j.

In particular, we consider extrinsic-update constraints ϵ_j that are inversely proportional to the Hamming weight of the jth column of $\mathbf{H}$ so that the net update per iteration has a maximum magnitude that is the same for each posterior value. The modified algorithm is referred to as the *variable-constraint TDMP algorithm*. The algorithm introduced in Chapter 6 is a special case of the variable-constraint algorithm in each all the constraints are the same (i.e., $\epsilon_j = \epsilon$ for all j); it is referred to as the *fixed-constraint TDMP algorithm*. The introduction of the constraints has a minimal impact on the execution time of the algorithm.

The benefit of the variable constraint for decoding the (2304,1152) WiMAX code is illustrated in Fig. 7.5, which shows the performance of the 8-bit variable-constraint MQ-TDMP algorithm with a resolution of 0.25 and $C_Q = 0.5$. The column weights of the code are 2, 3, and 6, and the corresponding constraints are 60, 40, and 20. The performance of the 8-bit variable-constraint algorithm is equal to the performance of 16-bit decoding for any probability of code-error of 10^{-4} or less. The average number of iterations differs between them by a small fraction of 1% for a given value of I_{max} and a given signal-to-noise ratio.

The performance of 8-bit variable-constraint decoding is shown in Fig. 7.6 for the optimal choice of variable constraints for the (16200,7200) DVB-S2 code. The column weights of the code are 2, 3, and 8, and the corresponding constraints are 60, 40, and 20. The resulting performance is within 0.05 dB of the performance of 16-bit decoding.

The 8-bit MQ-TDMP algorithm also exhibits reduced vulnerability to error in

the receiver's estimate of the signal-to-noise ratio if the variable-constraint technique is used than if the fixed-constraint technique is used [63]. Equivalently, it provides greater robustness against error in the gain-control normalization, G, of the received signal in Fig. 2.1.

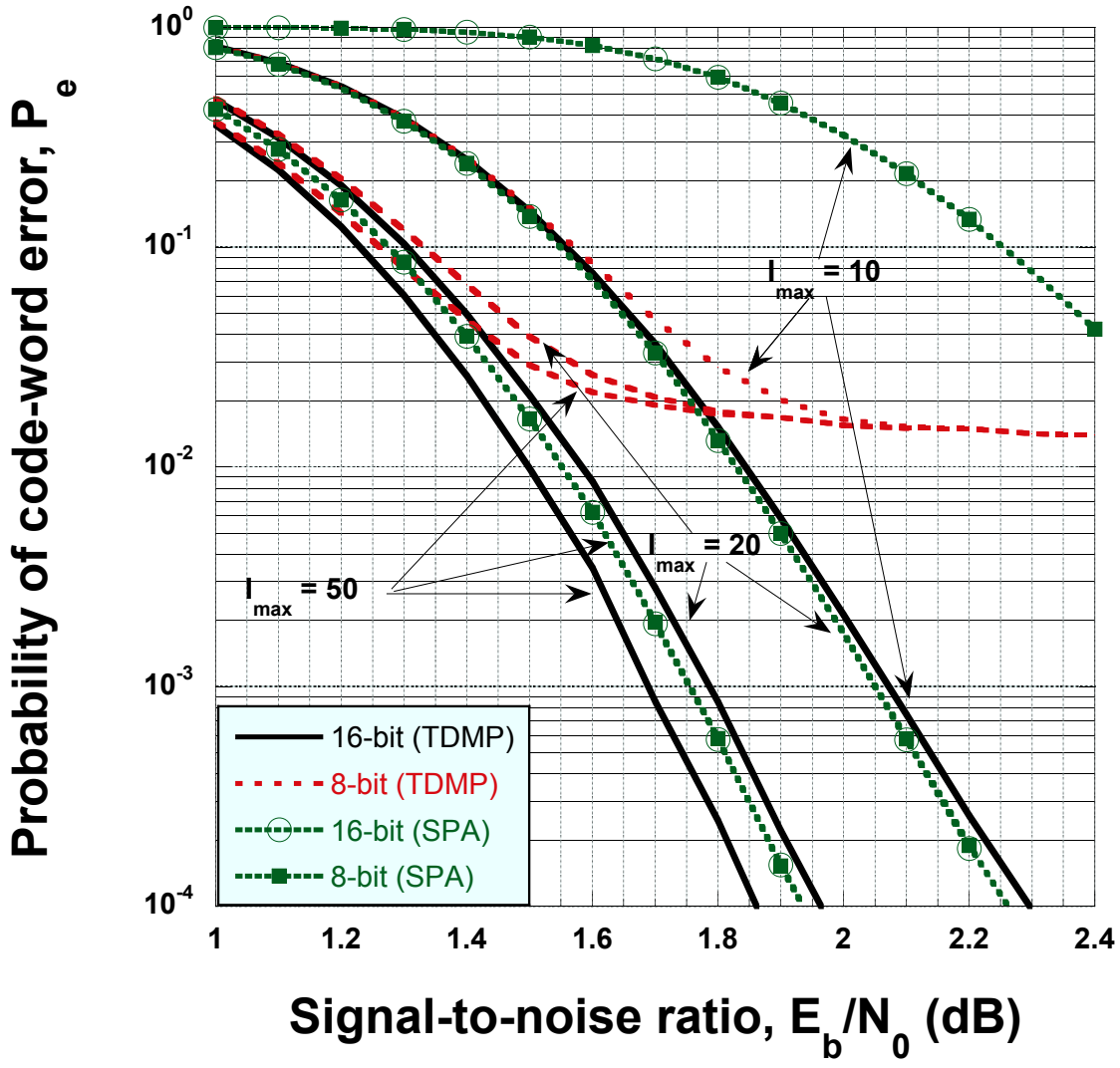

Figure 7.1: Comparison of fixed-point MQ-SPA and MQ-TDMP decoding.

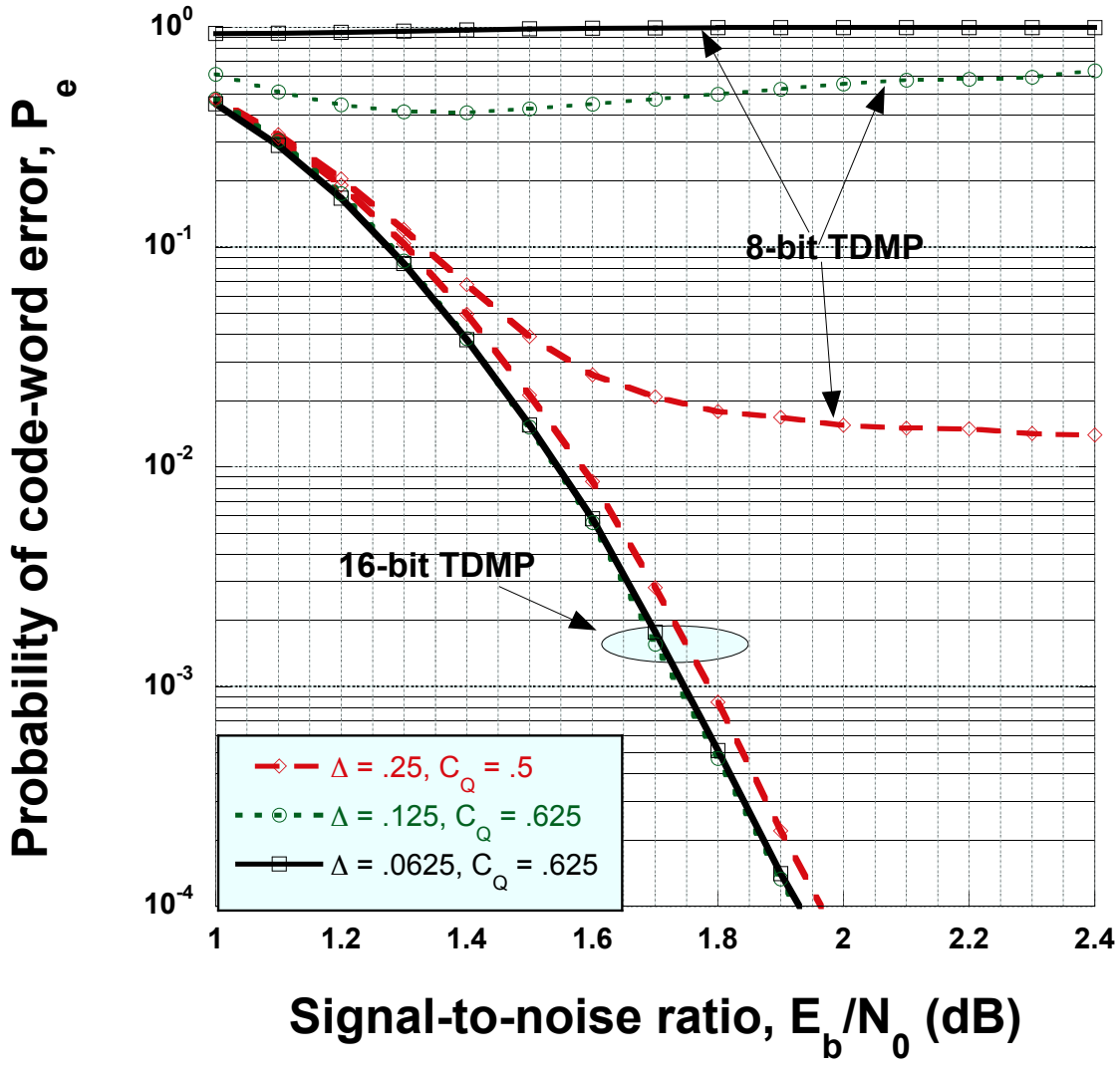

Figure 7.2: Performance of MQ-TDMP decoding with various decoder parameters.

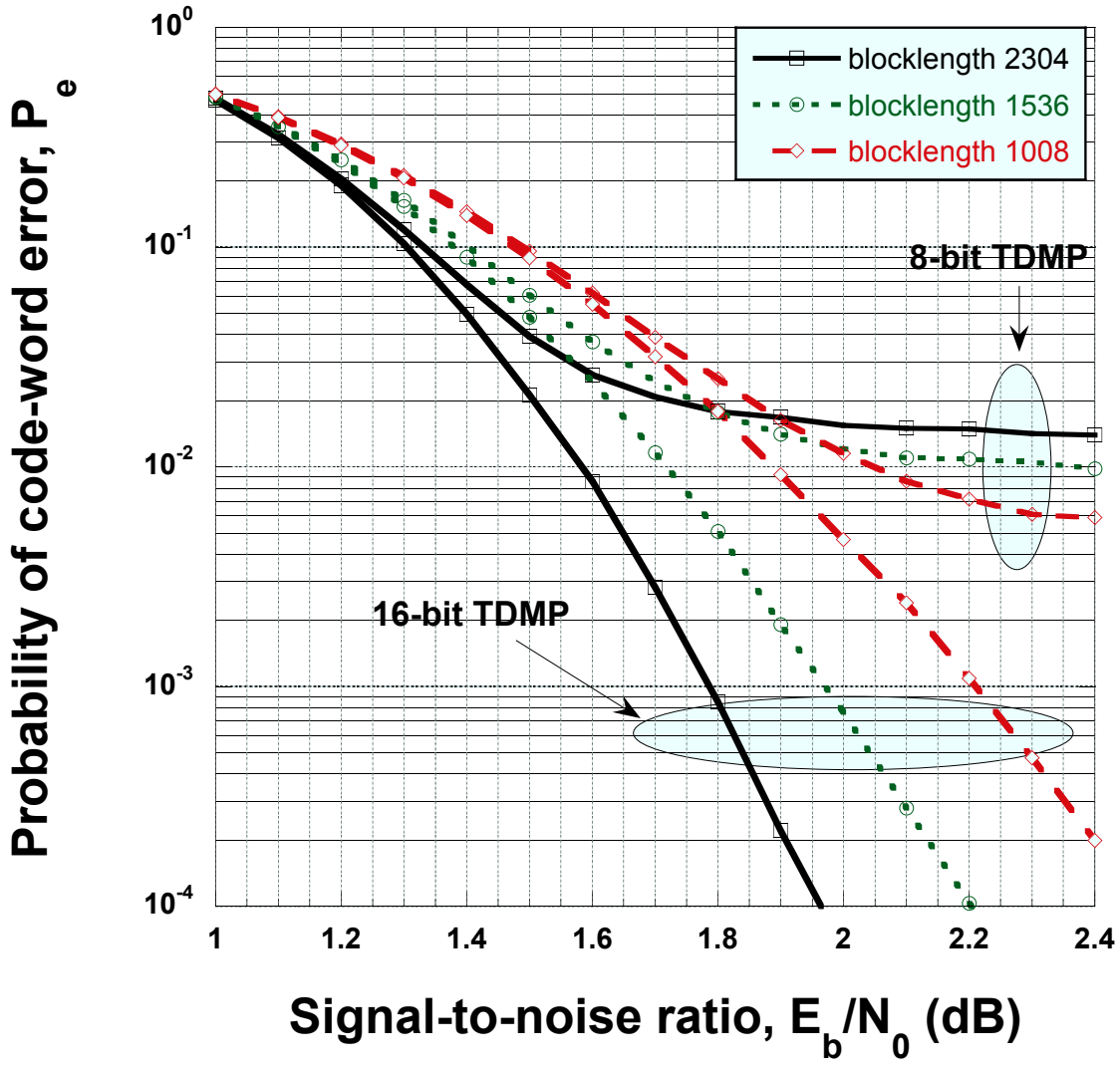

Figure 7.3: Performance of MQ-TDMP decoding with various block lengths.

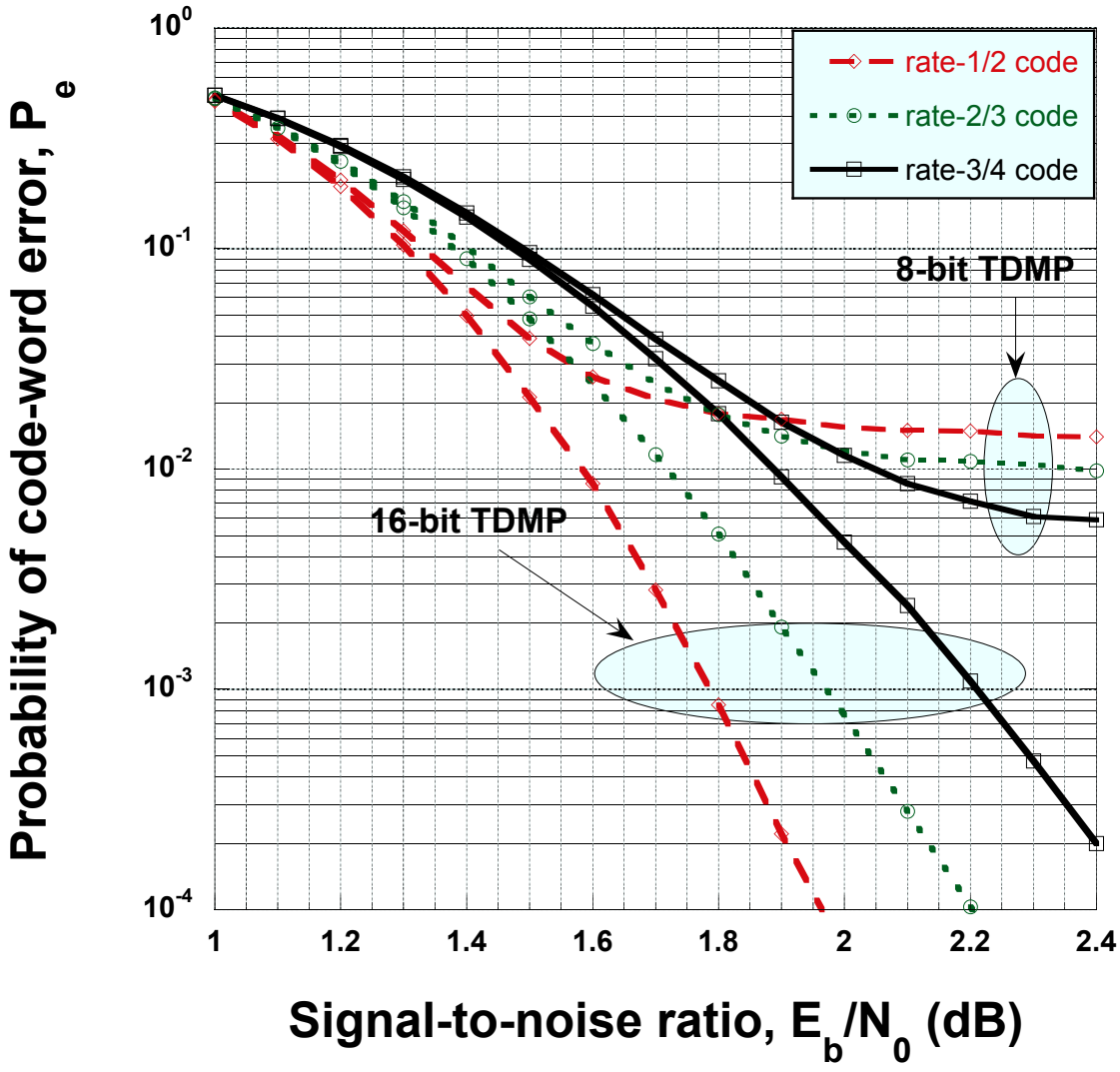

Figure 7.4: Performance of MQ-TDMP decoding with various code rates.

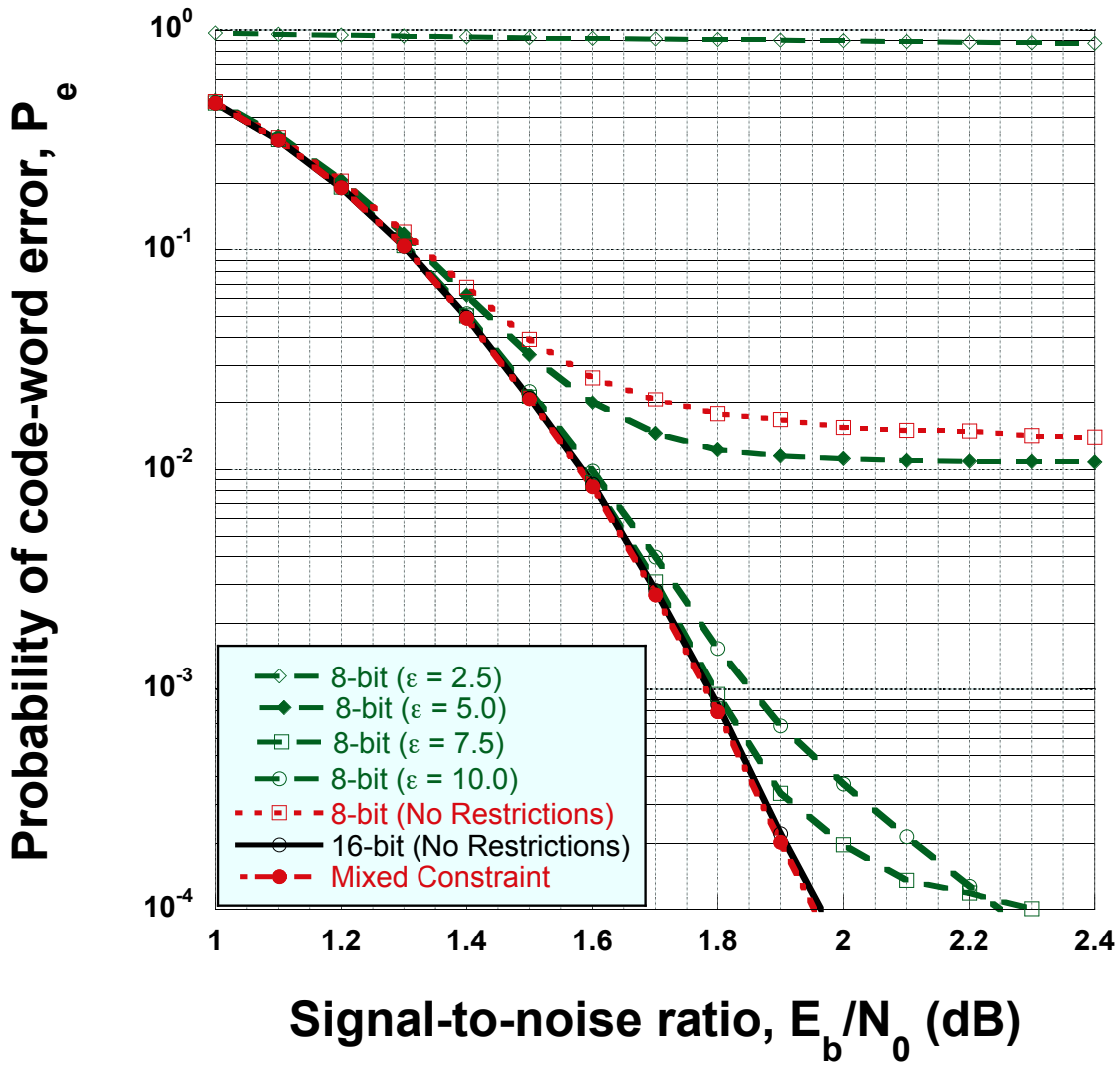

Figure 7.5: Performance of variable-constraint MQ-TDMP decoding for a WiMAX code.

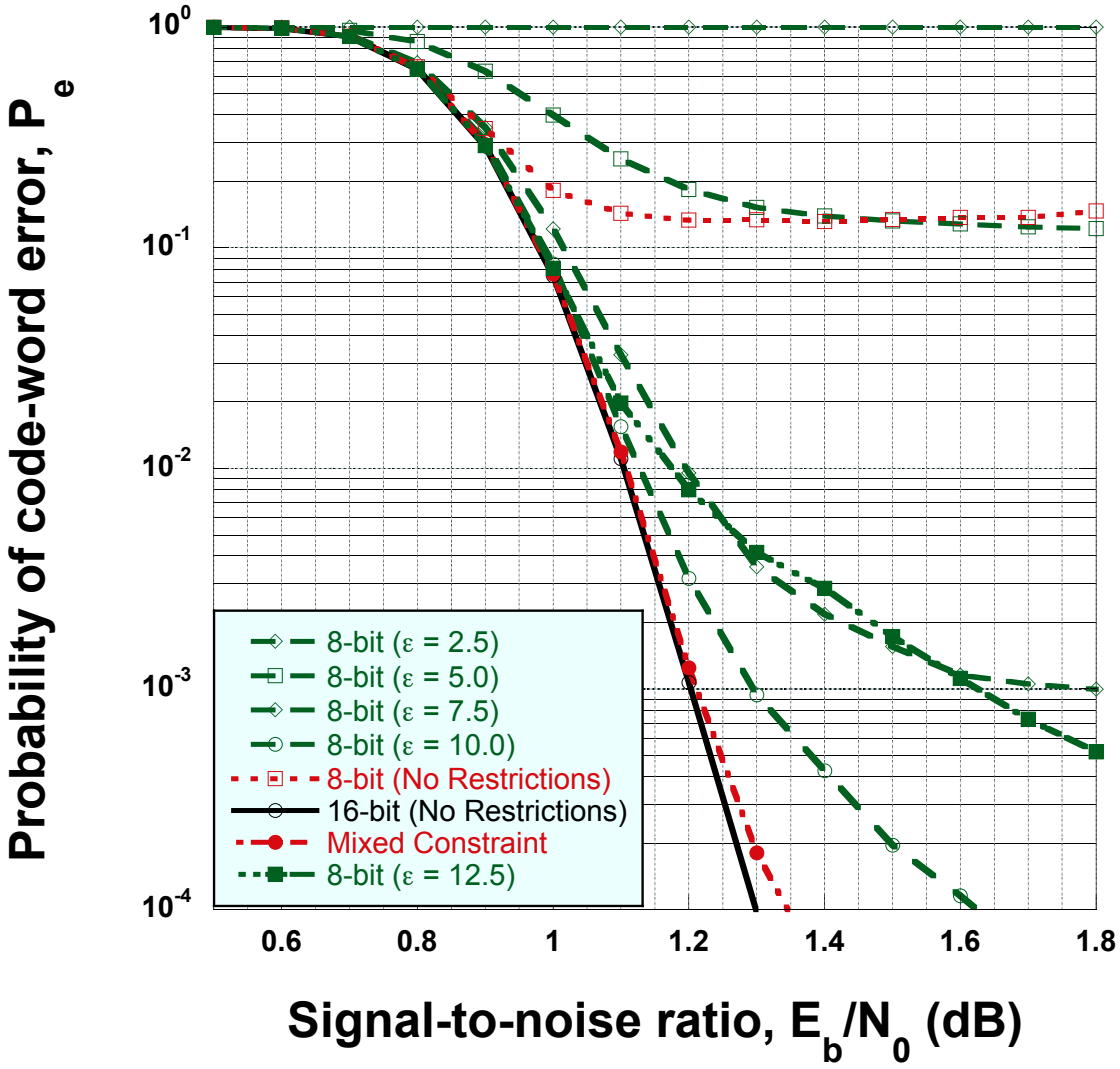

Figure 7.6: Performance of variable-constraint MQ-TDMP decoding for a DVB-S2 code.

Chapter 8

TDMP Schedules for Increased Throughput on a Stream Processor

Early termination can be achieved with the TDMP algorithm using the standard schedule, in which each iteration consists of a phase with message passing for all rows of the parity-check matrix followed by a phase with parity checks for all rows of the matrix. The algorithm guarantees that either the decoder produces a valid code word at termination or declares a decoder failure after the maximum allowed number of iterations. In this section, we consider three alternatives to the standard schedule for the offset-min-sum TDMP algorithm with early termination. Two of them are alternatives of practical interest that provide the same correctness guarantee as the standard schedule. The other alternative uses a naive approach that does not guarantee that the decoder's output is either a valid code word or a decoder failure. The decoder with the standard schedule and the decoder using the naive approach provide two benchmarks against which we evaluate the performance of the two practical alternatives to the standard schedule.

The performance of the algorithms is illustrated for four WiMAX-standard codes [7], each of which is a QC-LDPC code composed of circulant permutation submatrices of size 64. The block length of each code is 1536, and the rates of the codes are 1/2, 2/3, 3/4, and 5/6. (The rate-2/3 code and the rate-3/4 code are constructed using base-matrix option A in [7].)

The performance of OMS-TDMP decoding is evaluated for each code with the Storm-1 processor using its 8-bit mode for saturating, fixed-point arithmetic. The parameters of the decoding algorithm for each code are chosen to provide nearly-optimal performance with the standard schedule. In each instance, a resolution of $\Delta = 0.125$, an offset of $\eta = 0.125$, and an extrinsic-update magnitude constraint of $\epsilon = 2.5$ yield the lowest probability of error across signal-to-noise ratios of interest. A maximum of twenty decoder iterations is allowed for each received word, which is sufficient to provide most of the performance achievable with TDMP decoding for the four codes. Decoder execution times are evaluated with the processor decoding a sequence of consecutive received words as would be required in a practical communication system.

The decoding time per received word using the standard schedule is determined by the time required for the three components of decoding. The first component is the set-up and completion that includes the time required to initialize the DPU before the first iteration and the time required to recover the detected code word after the last iteration. The other two components represent the time for the message-passing phase of the TDMP algorithm per iteration and the time for the parity-check phase per iteration. The decoding time for a received word using the standard schedule is thus given by the set-up-and-completion time plus the number of decoding iterations multiplied by the sum of the message-passing time and the parity-check time for an iteration.

The first-row entry in Table 1 shows the processing times for the three decoding components for the rate-1/2 WiMAX code. (The entries in Table 1 are obtained from the cycle-accurate emulator of the Storm-1 processor and verified by decoder implementation on the processor.) The set-up and completion time is 2.13 μs. The message-passing phase for one iteration requires 2.05 μs, while the parity-check phase for one iteration requires 1.28 μs. Other than inter-lane communication of information during each phase, only a few operations are required per row in the parity-check phase, whereas many more operations are required in the message-passing phase. Yet the parity-check phase requires 62% of the time required for the message-passing phase, revealing that the processing time of the separate parity-check phase is dominated by the latency of exposed inter-lane communications.

The processing time per iteration can be reduced substantially if some means is employed to eliminate the inter-lane communications associated with performing parity checks. Consider an alternative approach to testing the correctness of hard decisions in which the parity checks for a given block of rows are incorporated within the corresponding message-passing subiterations. At the end of each subiteration of the message-passing phase, the posteriors updated during the subiteration are used to determine if the corresponding subset of parity checks are satisfied. The current posterior values that determine the hard decisions required for each parity check are already located in the stream-processor lane in which the parity check is performed since the updates of the same values have just been completed in the same lane. Thus the need for separate inter-lane communications for parity checks is eliminated. We refer to this non-standard schedule of parity checks as the *integrated parity check (IPC)*. The decoding time for the IPC is shown in the second-row entry in Table 1. The processing time for the integrated message-passing-and-parity-check phase of an iteration is only .08 μs greater than the message-passing phase alone in the standard

schedule.

First consider a decoder in which the IPC is used and decoding is terminated after an iteration in which the integrated parity checks are satisfied during all message-update subiterations. (There is no separate parity-check phase and thus no processing time allocated to such a phase.) The elimination of the parity-check phase in each iteration results in a substantial reduction in the processing time per iteration compared with decoding using the standard schedule. The algorithm does not guarantee that the detected word at termination (if any) is a valid code word, however; thus, we refer to it as the *naive IPC*.

The probability of error with TDMP decoding is much higher if the naive IPC is used than if the standard schedule is used. This is illustrated in Figure 8.1 in which the probability of code-word error is shown for both decoders (as well as two others discussed below) for the rate-1/2 WiMAX code. Decoding with the standard schedule results in a probability of code-word error of 10^{-3} at a signal-to-noise ratio of 1.97 dB, whereas acceptable performance is not achievable at a reasonable signal-to-noise ratio for decoding with the naive IPC. (Moreover, most code-word errors yield known decoder failures with the standard schedule, whereas the decoder with the naive IPC produces many outputs that are not valid code words but are not recognized as such by the decoder.) Similar results are observed when comparing the probability of bit error for the two decoders and when comparing the probability of error for the other three example codes. Thus decoding with the naive IPC is not of practical interest.

The validity of the detected word is guaranteed if the naive IPC is supplemented by a standard parity-check phase which is employed beginning with the first iteration in which all the integrated parity checks are satisfied. This modified IPC is referred to as the *IPC with confirmation*. Alternatively, the same validity guarantee is achieved if the naive IPC is modified by adding *stability checks* in each subiter-

ation. At the end of the subiteration, the lane performing the updates for a given row of the parity-check matrix determines if the updates have changed the sign of the posterior value for any variable node participating in the corresponding parity check. (I.e., it determines if the update has changed the tentative hard decision for any corresponding code symbol.)

The stability checks add .02 μs to the processing time for the message-passing phase of an iteration compared with the naive IPC and the IPC with confirmation, as shown in Table 1. If both the integrated parity checks and the stability checks are satisfied for each message-update subiteration in an iteration, the hard decisions at the end of the iteration are guaranteed to correspond to a valid code word (without the requirement of a separate parity-check phase). This modified IPC is referred to as the *IPC with stability check*.

As seen in Figure 8.1, both the IPC with confirmation and the IPC with stability check result in the same probability of code-word error as the standard schedule for the rate-1/2 WiMAX code. (For all three algorithms, almost all code-word errors are known decoder failures.) The probability of code-word error with each schedule is 10^{-3} if the signal-to-noise ratio is 1.97 dB, and it is 10^{-4} if the signal-to-noise ratio is 2.18 dB. Agreement of the error probabilities for the three schedules is also observed when considering the probability of bit error and when comparing the performance for the other three WiMAX codes.

The effect of each parity-check schedule on the decoder's information throughput is shown in Table 2 for the rate-1/2 WiMAX code. The throughput is measured as the average number of detected *information* bits per second. Decoding *without* early termination is considered for several values of the number of decoding iterations, I. (As noted above, the naive IPC is a decoder with a high error probability that provides a throughput benchmark for the practical IPC algorithms.) The stan-

dard schedule yields a decoder throughput of 87.4 Mbps and 11.2 Mbps respectively, for two and 20 iterations per code word. This represents 27% and 35% less decoder throughput, respectively, than with the naive IPC. The IPC with confirmation achieves a throughput of 100.2 Mbps with two decoder iterations and 16.7 Mbps with 20 decoder iterations. The throughputs are 17% less and 3% less than with the naive IPC for two and 20 iterations, respectively. (The IPC with confirmation is assumed to employ the parity-check phase only during the Ith iteration in the results shown in Table 2.) The IPC with stability check yields nearly the same throughput as the naive IPC for a given number of decoder iterations. (The difference is no more than 1% in each case shown in Table 2.)

The results in Table 2 do not reflect the fact that the different schedules result in a different average number of iterations if early termination is employed. Nor do they reflect the fact that the IPC with confirmation may initiate the parity-check phase in an iteration prior to the terminating iteration. The average number of decoding iterations for each of the three practical schedules with early termination is shown in Figure 8.2 as a function of the channel quality (measured by the probability of code-word error achievable by the decoder). For a channel which results in a probability of code-word error of 10^{-4} with each of the three schedules, the IPC with stability check requires an average of 6.1 iterations, whereas the standard schedule requires an average of only 5.1 iterations. Over the range of channel quality of practical interest, in fact, the average number of iterations required with the same two schedules differs consistently by about one iteration.

An average of 5.4 iterations are required by the IPC with confirmation for the channel which results in a probability of code-word error of 10^{-4}. This schedule requires approximately 0.2–0.3 iterations more than is required by the standard schedule on average for any channel quality of practical interest. It follows from the

definition of the algorithms that the naive IPC must result in a lower average number of iterations and higher throughput than either the IPC with confirmation or the IPC with stability check, but any other comparison of the throughputs of the schedules requires evaluation by execution on the processor or simulation in conjunction with Table 1. The average number of iterations required with each schedule approaches one asymptotically with improving channel quality, but the IPC with stability check approaches its limiting behavior more slowly than either the IPC with confirmation or the standard schedule.

The decoder's information throughput using each schedule with early termination is shown as a function of the signal-to-noise ratio in Figure 8.3 for the rate-1/2 WiMAX code. Both the IPC with confirmation and the IPC with stability check result in an information throughput of nearly 47 Mbps if the signal-to-noise ratio is 2 dB, which is 28% greater than the throughput achieved with the standard schedule. The throughput is 12% less than the throughput with the naive IPC, however, which reflects the extra processing cost of guaranteeing that the detected word at termination (if any) is a valid code word. The throughput improvement from using either the IPC with confirmation or the IPC with stability check in place of the standard schedule is reduced to about 21% if the signal-to-noise ratio is 2.8 dB, but the advantage of either over the standard schedule is substantial for signal-to-noise ratios of practical interest.

The IPC with confirmation provides a slightly greater throughput than the IPC with stability check with the rate-1/2 WiMAX code for signal-to-noise ratios between 1 dB and 5 dB. In the limit as the signal-to-noise ratio approaches infinity (in which each decoding attempt requires only one iteration), however, the throughput for both the standard schedule and the IPC with confirmation is approximately 140 Mbps, whereas the limiting throughput for both the naive IPC and the IPC with

stability check is nearly 180 Mbps. (Note that accurate simulated or emulated results for the average number of iterations can be obtained using a feasible number of sample outcomes at a higher signal-to-noise ratio than is feasible for accurately measuring the probability of code-word error.)

The information throughput for decoders using the standard schedule, the IPC with confirmation, and the IPC with stability check is shown in Table 3 for each of the four example WiMAX codes and two choices of the achievable probability of code-word error. Both the IPC with confirmation and the IPC with stability check provide much greater throughput than the standard schedule for the different rate WiMAX codes. The IPC with confirmation achieves 8.3%–30.0% greater throughput than the standard schedule over the four codes and the two channel conditions, whereas the throughput resulting with the IPC with stability check is 12.2%-26.0% greater than that obtained with the standard schedule. The largest percentage throughput improvements over the standard schedule are achieved with the lowest-rate code in the lower-quality channel, and the smallest percentage improvements occur with the highest-rate code and the higher-quality channel. The IPC with confirmation yields a slightly greater throughput than the IPC with stability check for the rate-1/2 and rate-3/4 codes, the two schedules result in the same throughput for the rate-2/3 code, and the IPC with stability check produces a slightly greater throughput than the IPC with confirmation for the rate-5/6 code.

Another characteristic of the decoder implementation which is of interest is the memory occupied by its executable code. The DPU in the Storm-1 processor includes an instruction memory of 2,048 384-bit instruction words. In our implementation, the rate-1/2 decoder using the IPC with stability check occupies only 21.2% of the instruction memory, whereas the rate-1/2 decoders using the standard schedule and the IPC with confirmation occupy 24.2% and 25.1% of the instruction memory,

respectively. The decoder using the IPC with stability check requires 23.0%, 28.4%, and 76.4% of the instruction memory for the codes of rates 2/3, 3/4, and 5/6, respectively. Over all four codes, 7%–21% more instruction memory is required if the standard schedule is used than if the IPC with stability check is used. Similarly, 10%–28% more instruction memory is required if the IPC with confirmation is used than if the IPC with stability check is used. The greater instruction-memory efficiency of the decoder using the IPC with stability check is the result of the absence of a separate parity-check phase in the schedule.

Table 8.1: Processing time of decoding components for the rate-1/2 WiMAX code.

Decoding algorithm	**Set-up & completion**	**Message passing only (per iter.)**	**Parity checks only (per iter.)**	**Integrated message passing & parity checks (per iter.)**
Standard schedule	2.13 μs	2.05 μs	1.28 μs	–
Naive IPC	2.13 μs	–	–	2.13 μs
IPC/confirmation	2.13 μs	–	1.28 μs	2.13 μs
IPC/stability check	2.13 μs	–	–	2.15 μs

Table 8.2: Information throughput without early termination for the rate-1/2 WiMAX code.

Decoding algorithm	$I=$ **2**	$I=$ **6**	$I=$ **10**	$I=$ **20**
Naive IPC (Mbps)	120.4	51.6	32.9	17.2
IPC w/ stability check (Mbps)	119.6	51.2	32.6	17.1
IPC w/ confirmation (Mbps)	100.2	47.5	31.1	16.7
Standard schedule (Mbps)	87.4	34.7	21.7	11.2

Table 8.3: Information throughput for the WiMAX code of each rate.

$P_{\mathbf{block}}$	$\mathbf{10^{-4}}$				$\mathbf{10^{-3}}$			
Rate	1/2	2/3	3/4	5/6	1/2	2/3	3/4	5/6
IPC w/ stability check (Mbps)	50.4	88.9	100.0	146.0	46.0	79.7	91.7	131.6
IPC w/ confirmation (Mbps)	51.3	88.9	101.0	140.9	46.8	79.7	93.0	128.9
Standard schedule (Mbps)	40.3	74.4	83.3	129.3	36.0	64.4	79.0	111.4

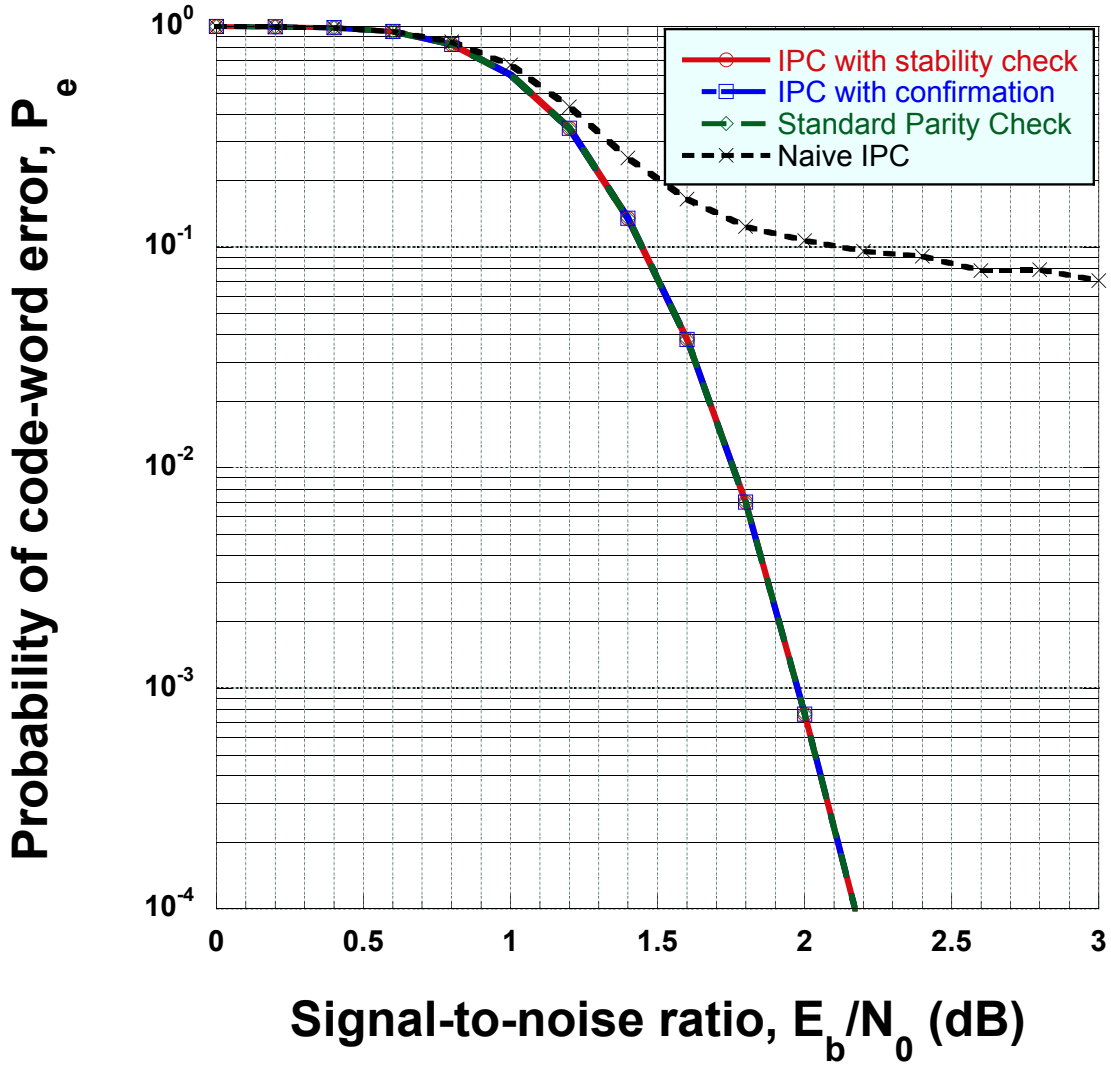

Figure 8.1: Probability of code-word error for the rate-1/2 WiMAX code.

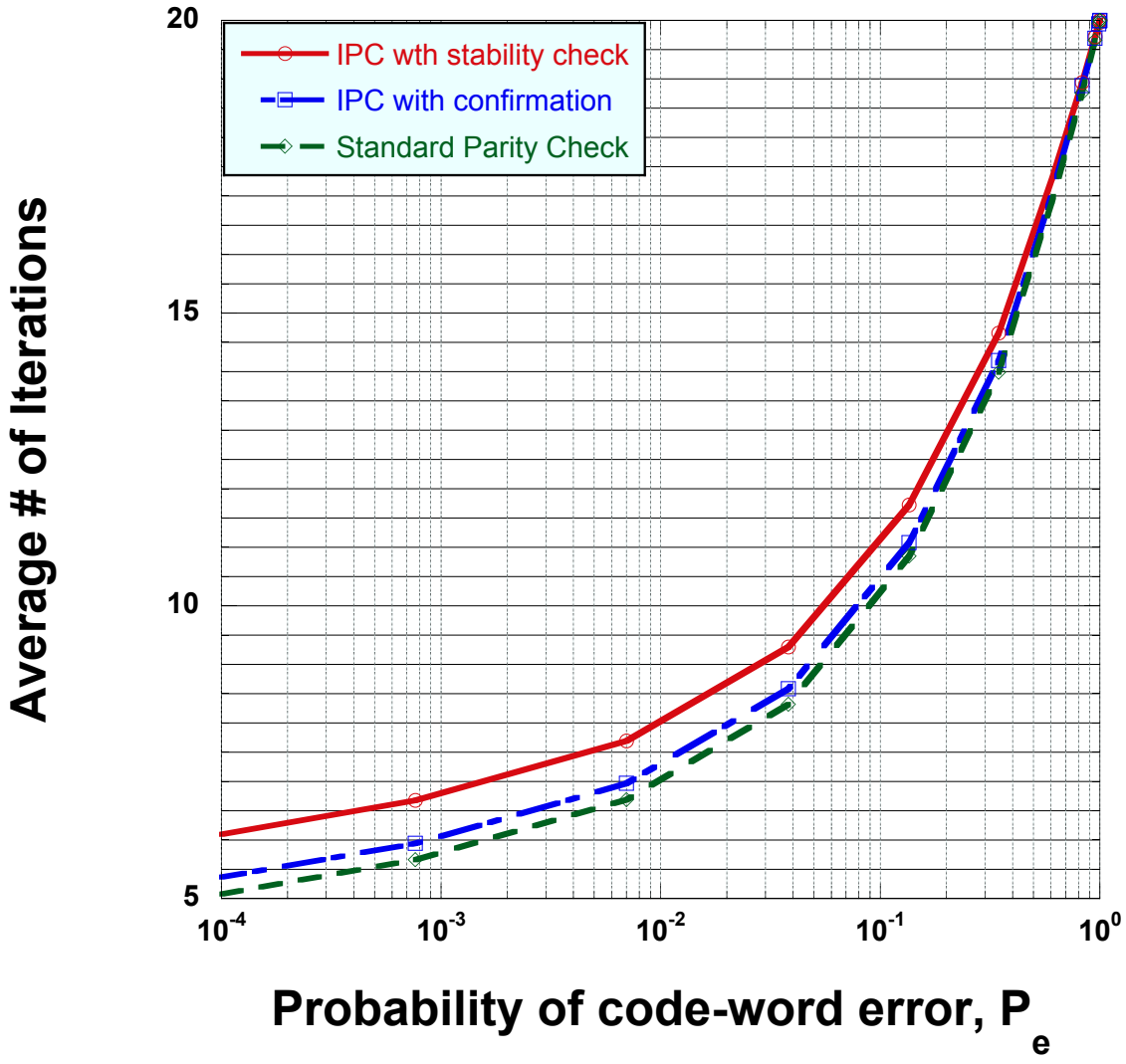

Figure 8.2: Average number of decoding iterations with early termination.

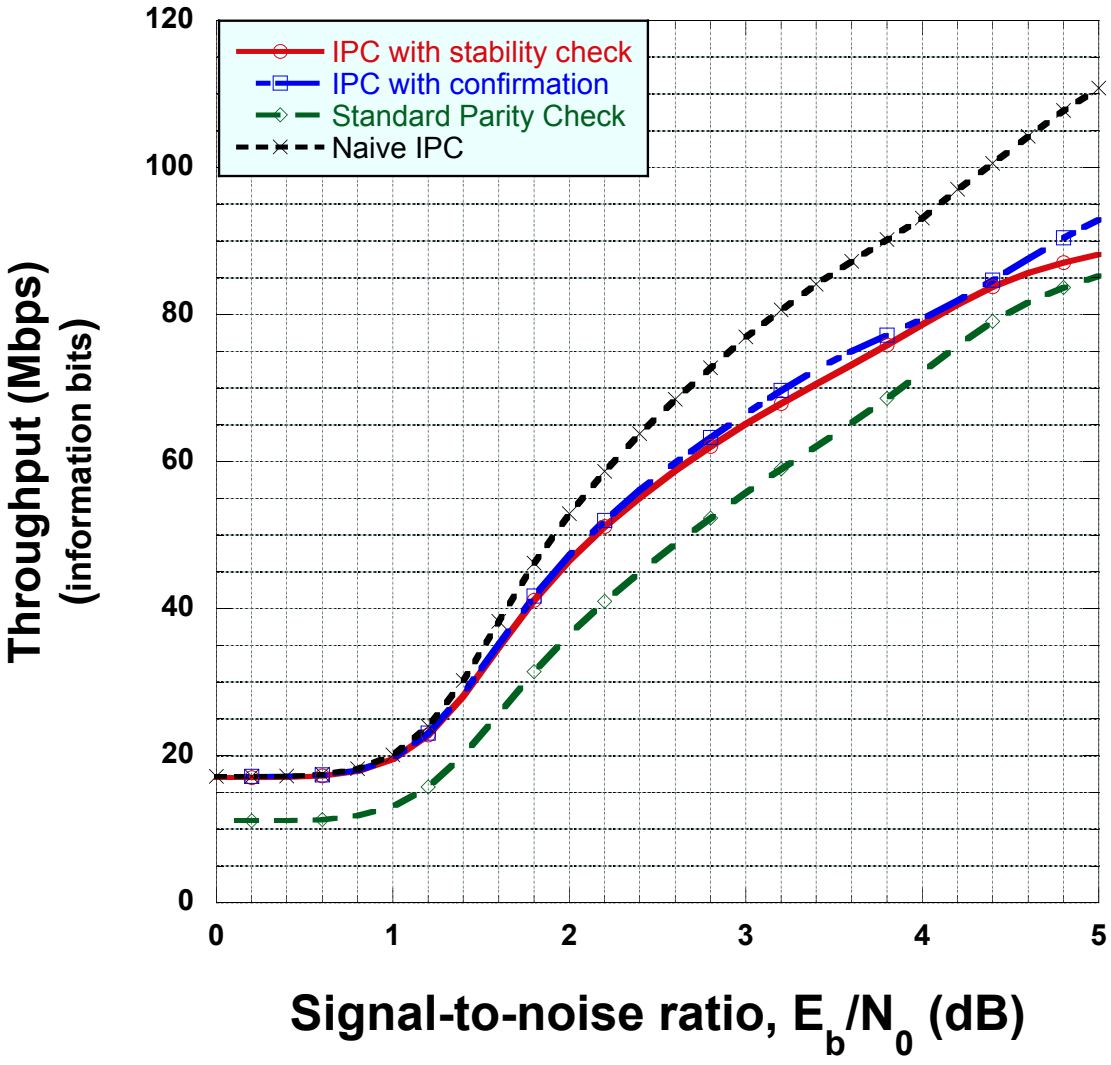

Figure 8.3: Information throughput with early-termination decoding.

Chapter 9

Conclusions

The performance of the TDMP decoding algorithm is much more sensitive to the precision of fixed-point saturating, signed arithmetic than is the performance of the SPA. The 8-bit precision available in many DSP architectures results in a substantial degradation in the performance of TDMP decoding for LDPC codes with a good error-correcting capability due to the frequent occurrence of arithmetic saturation, and the resulting performance is much poorer than the performance with 16-bit precision. The effect of saturation is mitigated by imposing a constraint on the maximum magnitude of any single extrinsic update of a code symbol's posterior value.

A properly chosen constraint in 8-bit decoding with the OMS-TDMP algorithm results in performance nearly equal to the performance of optimal 16-bit decoding. If the MQ extrinsic update is used instead of the OMS extrinsic update, the best performance is obtained only if the constraint depends on the degree of the variable node corresponding to the updated posterior value. Optimized variable update constraints permit the system to achieve the same performance with 8-bit MQ-TDMP

decoding as with 16-bit decoding. Both fixed-constraint 8-bit OMS-TDMP decoding and variable-constraint 8-bit MQ-TDMP decoding result in substantially lower decoding times than their 16-bit counterparts on packed-data, fixed-point SIMD processors, and both require only about one-half the decoding time on average of the SPA.

The DLP of a low-power, embedded stream processor can be used to accelerate the decoding of a QC-LDPC code using variants of the TDMP layered belief-propagation algorithm. The communications among functional-unit clusters of the processor is a significant factor in the decoding time using the standard approach of alternating message-passing and parity-check phases for TDMP decoding. An alternative approach in which the parity checks are integrated with the message-passing phase reduces the exposed communication latency within the processor. Either a separate parity-check confirmation phase or a check for stability of hard decisions throughout an iteration of the message-passing phase can be used to ensure a valid code word at termination of the modified algorithm. Both provide a substantial increase in the decoder throughput over the standard TDMP schedule at no cost in the error probability for decoding with early termination. The algorithm employing stability checks permits the most efficient use of instruction memory due to the absence of a separate parity-check phase.

Bibliography

[1] S. Lin and D. J. Costello Jr., Error control coding, 2nd ed., New Jersey: Pearson Prentice Hall, 2004.

[2] J. L. Hennessy and D. A. Patterson, Computer architecture: A quantitative approach, 4th ed., San Francisco: Morgan Kaufman, 2007.

[3] J. Goodacre and A. N. Sloss, "Parallelism and the ARM instruction set architecture," *IEEE Computer*, vol. 38, no. 7, pp. 42–50, July 2005.

[4] G. Blake, R. G. Dreslinski, and T. Mudge, "A survey of multicore processors: a review of their common attributes," *IEEE Signal Proc. Mag.*, vol. 26, no. 6, pp. 26–37, Nov. 2009.

[5] G. Lechner, J. Sayir, and M. Rupp, "Efficient DSP implementation of an LDPC decoder," in *Proc. IEEE Intl. Conf. Acoustics, Speech, Signal Proc.* (Montreal, Canada), vol. 4, May 2004, pp. 665–668.

[6] IEEE Computer Society, *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC), and Physical Layer (PHY) Specifications. Amendment 5: Enhancements for Higher Throughput*, IEEE Std. 802.11n-2009, 29 Oct. 2009.

[7] IEEE Computer Society and IEEE Microwave Theory and Techniques Society, *IEEE Standard for Local and metropolitan area networks. Part 16: Air Interface for Broadband Wireless Access Systems*, IEEE Std. 802.16-2009, 29 May 2009.

[8] European Telecommunications Standards Institute, *Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2)*, ETSI EN 302 307, V1.2.1, Apr. 2009.

[9] Y. S. Sun *et al.*, "Application-specific accelerators for communications," in Handbook of Signal Processing Systems (S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, eds.) , New York: Springer, 2010.

[10] C. H. Liu *et al.*, "An LDPC decoder chip based on self-routing network for IEEE 802.16e applications," *IEEE J. Solid-State Circuits*, vol. 43, no. 3, pp. 684–694, Mar. 2008.

[11] M. S. Khairy, M. M. Abdallah, and S. E.-D. Habib, "Efficient FPGA implementation of MIMO decoder for mobile WiMAX system," in *Proc. IEEE Intl. Conf. Commun.* (Dresden, Germany), June 2009.

[12] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks" in *Proc. Design Automation Conf.* (Las Vegas, NV), June 2001, pp. 684–689.

[13] M. Gomes *et al.*, "Serial LDPC decoding on a SIMD DSP using horizontal scheduling," in *Proc. 14th European Signal Proc. Conf.* (Florence, Italy), Sept. 2006, pp. 89–101.

[14] J. A. Kennedy and D. L. Noneaker, "Decoding of a quasi-cyclic LDPC code on a stream processor," in *Proc. IEEE Military Commun. Conf..* (San Jose, CA), Nov. 2010, pp. 2062–2067.

[15] B. Khailany *et al.*, "A programmable 512 GOPS stream processor for signal, image, and video processing," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 202–213, Jan. 2008.

[16] Y. Lin *et al.*, "SODA: A low-power architecture for software radio," in *Proc. Int. Symp. Comput. Architecture* (London, Canada), June 2006, pp. 89–101.

[17] M. Woh, "Architecture and analysis for next generation mobile signal processing," Ph.D. dissertation, University of Michigan, 2011.

[18] O. Anjum *et al.*, "State of the art baseband DSP platforms for software defined radio: A survey," *EURASIP J. Wireless Commun. Networking*, DOI:10.1186/1687-1499-2011-5, 2011.

[19] J. A. Kennedy and D. L. Noneaker, "Scheduling parity checks for increased throughput in early-termination, layered decoding of QC-LDPC codes on a stream processor," *EURASIP J. Wireless Commun. Networking: Special Issue on Algorithm and Implementation Aspects of Channel Codes and Iterative Receivers*, 2012,2012;41.

[20] M. M.. Mansour and N. R. Shanbhag, "High-throughput LDPC decoders," *IEEE Trans. Very Large Scale Integrated Systs.*, vol. 11, no. 6, pp. 976–996, Dec. 2003.

[21] D. E. Hocevar, "A reduced complexity decoder architecture via layerer decoding of LDPC codes," in *Proc. IEEE Workshop Signal Proc. Systs.* (Austin, TX), Oct. 2004, pp. 107–112.

[22] M. M. Mansour, "A turbo-decoding message-passing algorithm for sparse parity-check matrix codes," *IEEE Trans. Signal Proc. Systs.*, vol. 54, no. 11, pp. 4376–4392, Nov. 2006.

[23] J. Chen and M. P. C. Fossier, "Density evolution for two improved BP-based decoding algorithms of LDPC codes," *IEEE Commun. Ltrs.*, vol. 6, no. 5, pp. 208–210, May 2002.

[24] W. Dally *et al.*, "Stream processors: programmability with efficiency," *ACM Queue*, vol. 2, no. 1, pp. 52–62, Mar. 2004.

[25] U.J. Kapasi *et al.*, "Programmable stream processors," *IEEE Computer*, vol. 36, no. 8, pp. 54–62, Aug. 2003.

[26] J. Zhao, F. Zarkeshvari, and A. H. Banihashemi, "On implementation of min-sum algorithm and its modifications for decoding low-density parity-check (LDPC) codes," *IEEE Trans. Commun.*, vol. 53, no. 4, pp. 549–554, Apr. 2005.

[27] J. Chen *et al.*, "Reduced complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

[28] Z. Zhang *et al.*, "Design of LDPC decoders for improved low error rate performance: quantization and algorithm choices," *IEEE Trans. Wireless Commun.*, vol. 8, no. 11, pp. 3258–3268, Nov. 2009.

[29] D. Oh and K. K. Parhi, "Min-sum decoder architectures with reduced word length for LDPC codes," in *IEEE Trans. Circuits Systs. - I*, vol. 57, no. 1, pp. 105–115, Jan. 2010.

[30] C. Zhang *et al.*, "Flexible LDPC decoder design for multigigabit-per-second applications," in *IEEE Trans. Circuits Systs. - I*, vol. 57, no. 1, pp. 116–124, Jan. 2010.

[31] D. Kania and W. Sułek, "Code construction algorithm for architecture aware LDPC codes with low-error-floor," in *Proc. IEEE Region 8 Int. Conf. Comp. Technol. Electrical and Electronics Eng.* (Irkutsk Listvyanka, Russia), vol. 4, July 2008, pp. 1–6.

[32] M. M. Mansour and N. R. Shanbhag, "A 64-Mb/s 2048-bit programmable LDPC decoder chip," in *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 684–698, Mar. 2006.

[33] K. K. Abburi, "A scalable LDPC decoder on GPU," in *Proc. IEEE Intl. Conf. VLSI Design* (Chennai, India), Jan. 2011, pp. 183–187.

[34] D. Boa, B. Xiang, R. Shen, A. Pan, Y. Chen, and X. Y. Zeng, "Programmable architecture for Flexi-Mode QC-LDPC decoder supporting wireless LAN/MAN applications and beyond," in *IEEE Trans. Circuits Systs. - I*, vol. 57, no. 1, pp. 125–138, Jan. 2010.

[35] S. Huang *et al.*, "A flexible LDPC decoder architecture supporting two decoding algorithms," in *Proc. IEEE Int. Symp. Circuits Systs.* (Paris, France), June 2010, pp. 3929–3932.

[36] M. Rovini, G. Gentile, and L. Fanucci, "Fixed-point MAP decoding of channel codes," in *EURASIP J. Advances Signal Proc.*, DOI:10.1155/2011/184635, 2011.

[37] K. Gunnam, G. Choi, and W. Wang, M. Yeary, "Parallel VLSI architecture for layered decoding," *Texas A&M Technical Report*, May 2007.

[38] G. Falcão, L. Sousa, and V. Silva, "Massive parallel LDPC decoding on GPU," in *ACM Symp. Princ. Prac. Parallel Prog.* (Salt Lake City, UT), Feb. 2008, pp. 83–90.

[39] H. Ji, J. Cho, and W. Sung, "Massively parallel implementation of cyclic LDPC codes on a general purpose graphics processing unit," in *Proc. IEEE Workshop Signal Proc. Systs.* (Tampere, Finland), Oct. 2009, pp. 285–290.

[40] G. Wang *et al.*, "A massively parallel implementation of QC-LDPC decoder on GPU," in *IEEE Symp. Application Specific Processors* (San Diego, CA), June 2011, pp. 82–85.

[41] A. Gersho, "Quantization," *IEEE Commun. Mag.*, vol. 15, no. 5, pp. 16–29, Sept. 1977.

[42] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Ltrs.*, vol. 33, no. 6, pp. 457–458, March 1997.

[43] S. Y. Chung *et al.*, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Commun. Ltrs.*, vol. 5, no. 2, pp. 58–60, Feb. 2001.

[44] R. G. Gallager, "Low-density parity-check codes, *IRE Trans. Inform. Theory*, vol. IT-8, no. 1, pp. 21–28, Jan. 1962.

[45] R. Tanner, "A recursive approach to low complexity codes, *IEEE Trans. Inform. Theory*, vol. 27, no. 5, pp. 533–547, Sept. 1981.

[46] D. J. C. MacKay and R. M. Neal, "Good codes based on very sparse matrices," in *Proc. IMA Conf. Crypt. Coding* (Cirencester, UK), Dec. 1995, pp. 100–111.

[47] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1261–1271, Oct. 1996.

[48] T. Lestable *et al.*, "Block-LDPC codes vs duo-binary turbo-codes for European next generation wireless systems", in *Proc. Fall Vehicular Technol. Conf.* (Montreal, QC), Sept. 2006.

[49] Module 208: Telemetry Data Decoding, Rev. A, Deep Space Network Telecommunications Link Design Handbook, DSN 810-005/JPL D-19379, Jet Propulsion Laboratory, California Institute of Technology, May 2009.

[50] J. Zhang and M. P. C. Fossorier, "Shuffled iterative decoding," *IEEE Trans. Commun.*, vol. 53, no. 2, pp. 209–213, Feb. 2005.

[51] Z. Li *et al.*, "Efficient encoding of quasi-cyclic low-density parity-check codes, *IEEE Trans. Commun.*, vol. 54, no. 1, pp. 71–81, Jan. 2006.

[52] S. Myung, K. Yang, and J. Kim, "Quasi-cyclic LDPC codes for fast encoding, *IEEE Trans. Inform. Theory*, vol. 51, no. 8, pp. 2894–2901, Aug. 2005.

[53] R. M. Tanner, et al., "LDPC block and convolutional codes based on circulant matrices," *IEEE Trans. Inform. Theory*, vol. 50, no. 12, pp. 2966–2984, Dec. 2004.

[54] H. Jin, A. Khandekar, and R. J. McEliece, "Irregular repeat-accumulate codes", in *Proc. Int. Symp. Turbo Codes and Related Topics* (Brest, France), Sept. 2000, pp. 1–8.

[55] M. Eroz, F. W. Sun, and N. L. Lee, "DVB-S2 low density parity check codes with near Shannon limit performance, *Int. J. Satellite Commun. Networking*, vol. 22, no. 3, pp. 269–279, May/June 2004.

[56] S. Müller *et al.*, "A novel LDPC decoder for DVB-S2 IP," in *Proc. Design, Auto., Test in Europe* (Nice, France), Apr. 2009, pp. 1308–1313.

[57] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory* vol. 47, no. 2, pp. 498–519, Feb. 2001.

[58] E. Yeo *et al.*, "A flexible LDPC decoder architecture supporting two decoding algorithms," in *Proc. IEEE Int. Symp. Circuits Systs.* (San Antonio, TX), Nov. 2001, pp. 3019–3024.

[59] J. Dielissen, A. Hekstra, and V. Berg, "Low cost LDPC decoder for DVB-S2," in *Proc. Design, Auto., Test in Europe* (Munich, Germany), Mar. 2006, pp. 130–135.

[60] H. Lee, C. Chakrabarti, and T. Mudge, "A low-power DSP for wireless communications," *IEEE Trans. VLSI Syst.*, vol. 18, no. 9, pp. 1310–1322, Sept. 2010.

[61] W. Krimer *et al.*, "Synctium: A near-threshold stream processor for energy-constrained parallel applications," *IEEE Computer Architecture Letters*, vol. 9, no. 1, pp. 21–24, Jan.-June 2010.

[62] W. J. Dally *et al.*, "Efficient embedded computing," *IEEE Computer*, vol. 41, no. 7, pp. 27–32, July 2008.

[63] J. A. Kennedy and D. L. Noneaker, "A technique to improve the performance of fixed-point TDMP decoding of QC-LDPC codes in the presence of SNR estimation error," in *Proc. IEEE Military Commun. Conf.*. (Baltimore, MD), Nov. 2011, pp. 649–654.